IEEE MICRO

32-bit microprocessors

Industry's sweet gift to itself

DECEMBER 1985

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

CALL FOR PAPERS

A symposium sponsored by

IEEE Computer Society
Technical Committee on
Computer Communications

computer networking symposium

Conference Chairperson
Tuncay Saydam
University of Delaware

Technical Program Chairperson
George Chang
Bell Communications Research

Program Committee
Robert Klessig
Bell Communications Research
Ravi Mazumdar
Columbia University
Adarshpal Sethi
University of Delaware
Pravin Varaiya
U.C. Berkeley

Tutorial Chairperson
Guy Omidyar
Magnavox Inc.

WASHINGTON, D.C. November 17-18, 1986

You are invited to submit a technical paper for presentation at the 1986 Computer Networking Symposium sponsored by the IEEE Computer Society Technical Committee on Computer Communications.

The symposium will highlight papers dealing with topics related to the design, modeling, selection, performance, and implementation of current and soon to be available network systems.

All papers will be reviewed by the Program Committee and authors of accepted papers are expected to present them on November 18, 1986 at the conference in Washington, D.C. Papers accepted for presentation will be published in the Symposium Proceedings.

Topics of Interest include, but are not limited to:

- Long haul networks
- Local area networks
- PBX systems
- Satellite systems
- Video systems
- Protocols
- Teleconferencing
- Standards design
- Network testing
- Network procurements
- Internetworking

Author's Schedule

April 30, 1986 for Papers
Submit 4 copies of your paper or a 1,000 word extended abstract to:
Computer Networking Symposium
1730 Massachusetts Ave., N.W.
Washington, D.C. 20036-1903

June 30, 1986
Notification of provisional acceptance

July 31, 1986
Papers due in final form

November 18, 1986
Presentation at the Symposium

☐ Please include my name for further information on the Computer Networking Symposium

Name _____

Organization _____

Address _____

City _____

State _____ Zip _____

Please INCLUDE your return address and phone number

Phone number _____

Send to: **COMPUTER NETWORKING**
1730 Massachusetts Ave, NW
Washington, DC 20036-1903

IEEE COMPUTER SOCIETY

IEEE THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE MICRO

Volume 5 Number 6 (ISSN 0272-1732) December 1985

On the cover

'Tis the season for gifts—and gifts for the marketplace such as the long-awaited 32-bit microprocessor can bring sweet thoughts to a computer industry contemplating new revenues.

Cover: Jay Simpson and Larry Keiser

Features

Special Report

MicroStandards Special Features

Departments

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society: IEEE Headquarters, 345 East 47th St., New York, NY 10017; IEEE Computer Society West Coast Office, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578. Annual subscription: $12.00 in addition to IEEE Computer Society or any other IEEE society member dues. Nonmember prices: available on request. Single-copy prices: members $7.50; nonmembers $15.00. This journal is also available in microfiche form.

Undelivered copies: Send to 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.

Postmaster: Send address changes to *IEEE Micro,* IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854. Second class postage is paid at New York, NY, and at additional mailing offices.

From the Editor-in-Chief

First, I need to thank our Associate Editor-in-Chief, Dr. Joe Hootman, and several members of the *IEEE Micro* Editorial Board for their efforts in getting the December manuscripts reviewed. Of course, our staff people, Marie English and Joe Schallan, did their usual excellent job.

I will continue to be commuting between Texas and Arizona for another month, so please tolerate any tardiness on my part in responding to your correspondence. My new mailing address for manuscript submittal, letters to the Editor, or other correspondence is:

James J. Farrell III
VLSI Technology Inc.
I0220 South 5Ist Street
Phoenix, AZ 85044

In late October the Computer Society's Editors-in-Chief met in Yorktown, NY, with John Musa, our VP of Publications, and True Seaborn, our publisher. We discussed several topics, but the major one was our 1986 budget for publications. The recession being experienced in the computer and semiconductor industries is also being felt in the Computer Society. To help make our 1986 expenditures more nearly equal our expected income, we recommended several actions.

Some suggestions were: deferral of additional pages for *Computer* magazine, tight page budgets for all the magazines, hiring and equipment-purchase postponements, and other similiar measures. However, we all firmly agreed that the quality of the magazine would not be compromised.

More than half of the correspondence that I received from readers since the last issue has been from countries other than the USA. *IEEE Micro* is truly becoming an international magazine. In response to this I have made a concerted effort to solicit international candidates for open Editorial Board positions. You will see some results by the next issue.

To give you some insight into the comments readers have been sending me, I list a synopsis of them:

"Reviews and product updates are late. . .monthlies and weeklies do better." C.H.B., Tucker, GA

"Disliked MicroLaw. . .where is piracy info?" G.K.G., Port Angeles, WA

"Liked Futurebus. . . . .disliked nontechnical junk on office terminals and politics." B.M., Bellvue, NE

"Liked 'A Signal Processing Implementation for IBM PC'. . ." R.L.M., Lynchburg, VA

"(Format) fine, go on!" K.W. Switzerland

"Liked 'FERMTOR'. . .Mapping High-Level Syntax. . ." L.F.P., Santiago, Chile

"Would like more practical design. . ." A.I.T., Buenos Aires, Argentina

"Liked 'Structure for Computer Networking'. . ." N.N.K., Pilani, India

"Liked study of IEEE 796 bus multiprocessor. . ." N.D.J., Aberdeen, Scotland, UK

"Justify both sides of the (printed) column. . ." G.B., Portola Valley, CA

"Would like to see the latest on microcomputer chips. . ." S.S., NY, NY

"'Second-sourcing CPUs' was great. . ." K.A., Switzerland

"*Micro* is one of my favorites." A.D.W., Lewisburg, PA

"I liked the OLD (magazine) format." R.H.M., Rolla, MO

"I would like overviews at the beginning of the articles. . ." K.S.A., Tamil, India

"I would like more tutorials about microprocessors and I/O devices." O.A.A., Lima, Peru

"Future topics look interesting. . .advertise them in CS press." L.C., Torino, Italy

"Liked 'FERMTOR'. . ." W.V.M., Naperville, IL

Thank you for your comments. Please keep the cards and letters coming. They are our report cards.

Best Wishes,

Jim Farrell

The Intel 80386—Architecture and Implementation

Khaled A. El-Ayat and Rakesh K. Agarwal
Intel Corporation

The Intel 80386 represents the state of the art in high-performance, 32-bit microprocessors. It features absolute object code compatibility with previous members of the iAPX 86 family of microprocessors, including the 80286, 80186, 80188, 8086, and 8088. This protects major investments in application and operating systems software developed for the iAPX 86 family, while offering a significant enhancement in performance. The 80386's architecture and performance should allow it to be used in a wide range of demanding applications—e.g., in engineering workstations, office systems, robotic and control systems, and expert systems.

The 80386 implements a full 32-bit architecture with a 32-bit-wide internal data path including registers, ALU, and internal buses; it provides 32-bit instructions, addressing capability, and data types, and a 32-bit external bus interface. It extends the iAPX 86 family architecture with additional instructions, addressing modes, and data types. It incorporates a complete memory management unit. The 80386 extends the 80286 segmentation model to support four-gigabyte segments and to provide a standard two-level paging mechanism for physical memory management. System designers can use segmentation or paging or both, without performance penalties, to meet their memory management requirements.

The 80386 architecture is complemented by a bus interface that uses only two clocks per bus cycle; this allows efficient interfacing to high-speed as well as low-speed memory systems. At 16 MHz, the bus can sustain a 32-megabyte-per-second transfer rate. Other bus features include dynamic bus sizing to support mixed 16/32-bit port interfacing and a dynamically selectable pipelined mode to facilitate high-speed memory interleaving and allow longer access times.

The 80386 is implemented in Intel's CHMOS-III 1.5-micrometer process. Typical instruction mixes indicate an average processing rate of 4.4 clocks per instruction and an overall execution rate of three to four MIPS. To facilitate system debugging, the chip incorporates hardware debug features and self-testing.

80386 base architecture

Different microprocessor applications require different types of architectural support. Some applications—such as those running under Berkeley UNIX—may prefer a linear address space. Others that manage a multitude of dynamic data structures may require hardware-enforced rules to protect the visibility of the dynamically created objects. The 80386 architecture supports these diverse require-

0272-1732/85/1200-0004$01.00 © 1985 IEEE

ments by providing the user with several memory management and addressing models. Further, its repertoire of addressing modes, data types, instructions, and special constructs make it well suited to modern high-level languages.

The base architecture of the 80386 encompasses the register model, data types, addressing modes, and instruction set. It forms the basis for high-level-language compiler code generation and for assembly-language-level application programming. Other features of the machine useful for implementing operating systems are discussed in the section on OS architecture.

Registers. The 80386 possesses several on-chip register sets to support various machine features. Figure 1 shows the eight general-purpose registers available for calculations and memory addressing, the flags register, and the instruction pointer. Other registers include control registers, six segment registers used to structure the four-gigabyte address space and to facilitate system debug, and six debug registers used to control the setting of up to four code or data breakpoints.

The 32-bit general registers are named EAX, EBX, ECX, EDX, ESP, EBP, ESI, and EDI. To allow 16-bit operations and to provide compatibility with the 16-bit members of the iAPX 86 family, eight 16-bit registers are superimposed onto the low-order parts of the 32-bit registers. Similarly, there are eight 8-bit registers that are aliases for the lower and upper halves of each of the 16-bit registers. Operations on 8-bit or 16-bit registers affect only the corresponding superimposed registers. For example, the carry out of bit 7 during an 8-bit add is not propagated into bit 9 of the destination; instead, the carry flag (CF) of the flags register is set appropriately. This is true for all condition code settings in the flags.

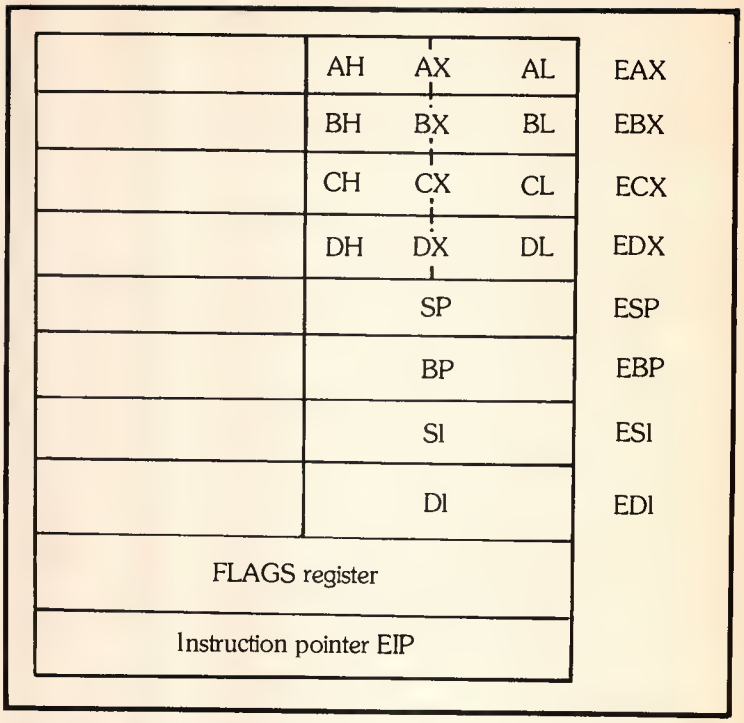


Figure 1. The 80386 general register set, FLAGS, and instruction pointer.

Operand addressing. 80386 operands may reside on the chip (in registers), in main memory, or in the I/O address space. Furthermore, an operand may be implied in the instruction or specified explicitly as a part of the instruction.

Storing operands in registers generally provides the fastest method of processing data. The contents of any 80386 general register can be operated on by any arithmetic or logical operator. Alternatively, 8-, 16-, or 32-bit constants (immediates) can be embedded directly in an instruction. Sixteen- and 32-bit operations may specify 8-bit sign-extended or zero-extended immediates. Table 1 includes sample instructions employing registers and immediates as operands. In general, register-to-register operations execute in two clocks on the 80386. At a clock rate of 16 MHz, this translates to 125 nanoseconds per operation.

Most operands are stored in main memory. The 80386 has a full complement of address generation mechanisms for specifying the effective address of such operands. These mechanisms were developed in response to the storage paradigms present in high-level languages.

In its simplest form, the effective address of a memory operand can be encoded directly in an instruction. Usually, however, a particular memory address is not known until the program is actually executing. In this case, the effective address can be obtained by summing the contents of one or two general-purpose registers and an optional immediate value or *displacement*. This register-based effective address scheme can be summarized as

[base register] + [index register] * (scale) + [displacement].

Here the base register is any general-purpose register and the index register is any general-purpose register

Table 1.
Examples of operand addressing in the 80386.

Instruction	Clocks	Semantics
INC EAX	2	Increment contents of EAX by 1.
IMUL EBX, -3	9	Multiply the integer in EBX by -3.
CMP CX, 0	2	Compare contents of CX with 0 and set condition codes.
MOVSX EAX, SI	3	Sign extend the contents of the 16-bit register SI and move into EAX.
MOV DWORD PTR [56], -12445654	2	Assign -12445654 to the 32-bit integer at address 56.
JMP jumpTable[EBX*4]	10	Jump to the address stored at entry EBX of jump table.
SUB DX, WORD PTR [EBP + EDI*2-10]	7	Subtract from DX the 16-bit quantity at address [EBP + EDI*2-10].

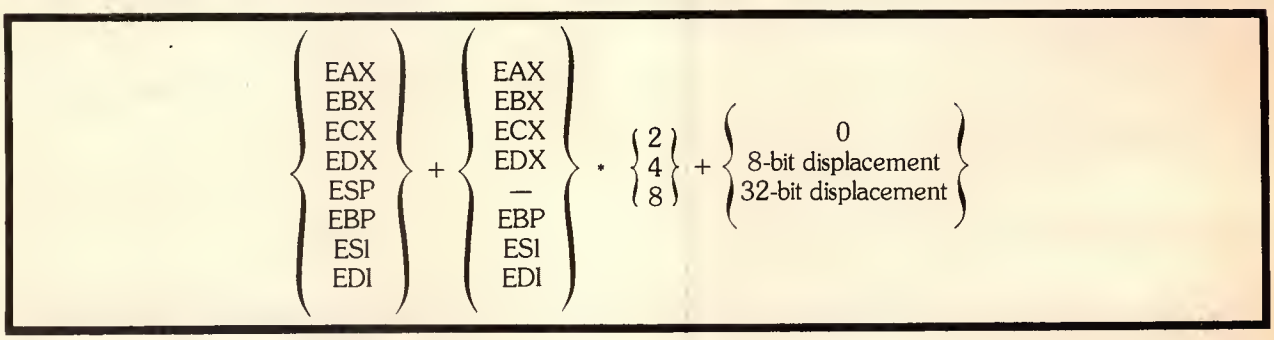


Figure 2. 32-bit memory addressing modes.

Table 2.
80386 support of high-level-language memory addressing.

Storage class	Type specifier	Addressing mode
Static	Scalar	[disp]
	Structure	[disp]
	Array of scalars	[disp + index]
	Array of structures	[disp + index]
Automatic	Scalar	[base + disp]
	Structure	[base + disp]
	Array of scalars	[base + disp + index]
	Array of structures	[base + disp + index]
Heap	Scalar	[base]
	Structure	[base + disp]
	Array of scalars	[base + index]
	Array of structures	[base + disp + scale]

other than ESP. The scale specification is a constant value, either 2, 4, or 8. If specified, it scales the index register by the required amount, thus simplifying indexing into arrays of multibyte elements. The displacement field is also a constant, its value ranging from -2^{31} to $+2^{31}-1$. Figure 2 shows all of the 80386's 32-bit memory addressing modes, and Table 2 correlates high-level-language addressing forms with those modes. (Example of memory addressing appear in Table 1.)

Data types. As shown in Figure 3, the 80386 directly supports the fundamental data types found in most high-level languages. The basic operations provided by the 80386 for each of these data types are shown in Table 3. Most of these operations execute in two clocks when register or immediate operands are used. Furthermore, because of pipelining and the two-clock memory bus, stores to memory also execute in two clocks.

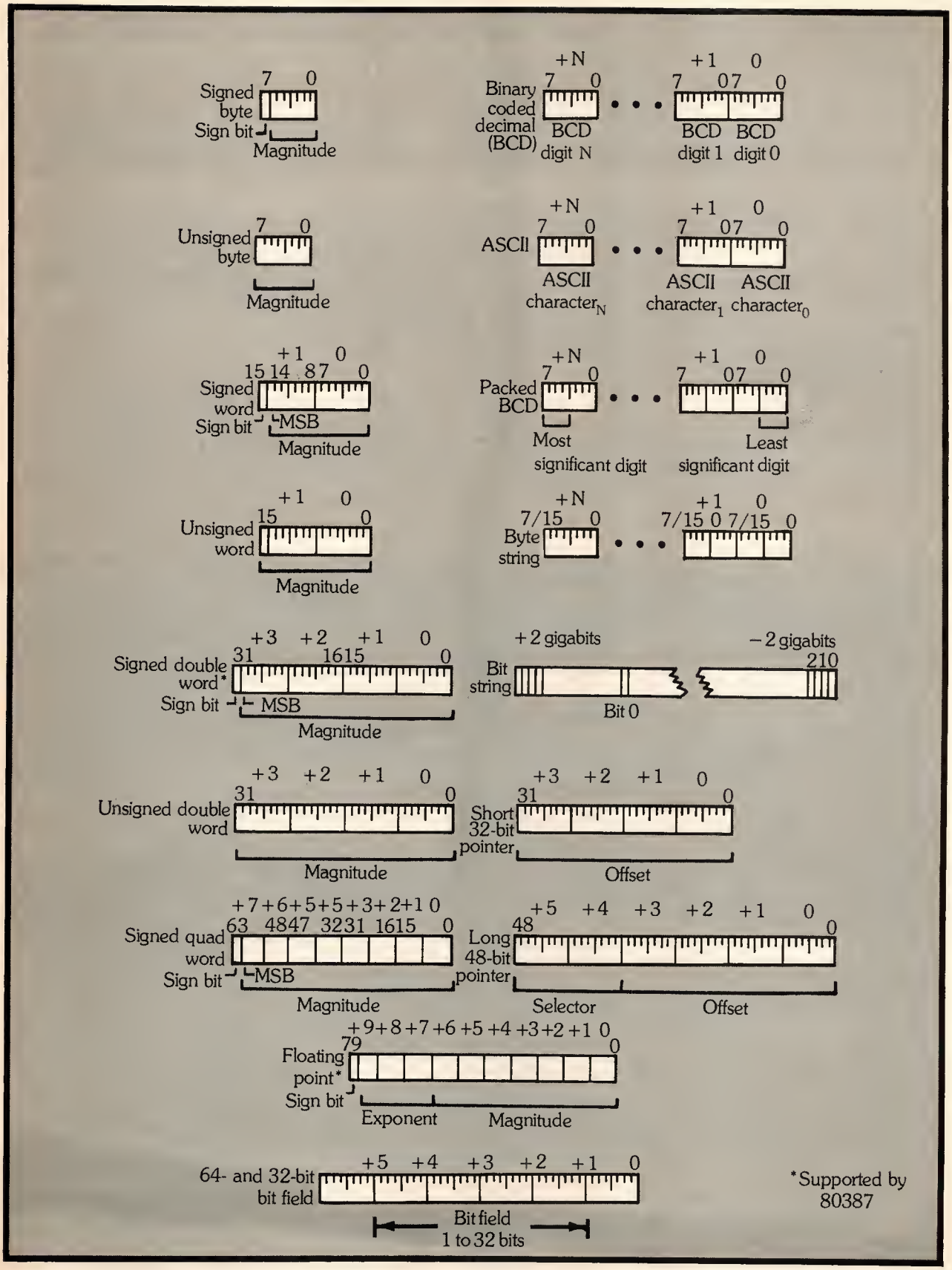


Figure 3. 80386 data types.

The basic unit of storage is a *byte*; a 16-bit quantity is a *word*, and a 32-bit quantity is a *double word*, or *d-word*. Words are defined as having a length of 16 bits so that notational compatibility with the other members of the iAPX 86 processor family will be retained. In the 80386, most data types are represented in the form of bytes, words, or d-words, or combinations thereof.

Words comprise two consecutive bytes in memory, with the low-order byte at the lower-numbered address. D-words comprise four consecutive bytes in memory, with the low-order byte at the lowest address and the high-order byte at the highest address. The address of a word or d-word is the address of the low-order byte. Hence, the 80386 utilizes the little-endian storage scheme.

Ordinal. An ordinal is an unsigned number. If it is in the range 0 through 4,294,967,295, it corresponds to a d-word value. If it has a magnitude of less than zero, it corresponds to a word or byte value. An example of an ordinal operation is the instruction sequence

```
MUL   EBX,vec[EDX*4] ; EBX := EBX * vec[EDX]
INTO                 ; Generate an exception if
                     ;overflow
```

Here, the content of EBX is multiplied by the EDXth element of the d-word-sized ordinal array vec, and the product is stored in EBX. An overflow exception is generated if the product exceeds 4,294,967,295.

Integer. An integer is a signed number in the range −2,147,483,648 through +2,147,483,647. As with ordinals, d-word, word, and byte integers are supported. Integers are represented in two's-complement

80386 system debug capabilities

A large portion of system development time is usually devoted to system debugging and verification. The magnitude of the problem is strongly influenced by system complexity at both the software and the hardware levels. In highly complex systems, external hardware and software debug aids alone cannot provide the level of support needed; internal CPU assistance is required.

To facilitate system development and real-time system debugging, the 80386 provides the following capabilities:

- detection of instruction breakpoints,
- detection of data reference breakpoints,
- specification of four separate breakpoint addresses,
- instruction single-stepping, and
- a one-byte trap instruction.

The 80386 has six system debug registers (see figure). The first four, DR0 to DR3, store the required breakpoint addresses. Registers DR6 and DR7 contain debug status and control information, respectively. Registers DR4 and DR5 are reserved by Intel. Breakpoint addresses must be linear addresses of instructions or data items. The control register, DR7, specifies the conditions under which a breakpoint is recognized and includes enable/disable masking fields, the breakpoint type, and the breakpoint length fields.

The enable/disable masking fields determine whether a detected breakpoint condition will be recognized by the CPU and whether an exception will be generated or simply stored in the debug status register for future examination. The breakpoint type field indicates the type of memory reference—e.g., an instruction execution, a data write reference, or a data read/write reference—that is intended to cause the system break. The breakpoint length field is used primarily for data references and selects byte, word, or double-word ranges for data item breakpoints. This field is needed because of a problem that arises in data referencing. Simply specifying the starting address of a data item is too restrictive and is insufficient for matching a breakpoint condition. The problem exists because there are three different data item lengths (8, 16, and 32); under erroneous conditions the generated address and data type length may not exactly match the specified breakpoint condition. The length field adds flexibility by selecting a range in which breakpoints can occur. Instruction breakpoints always specify a one-byte length field, since system breakpoints should uniquely specify the byte-granular starting address of intended instructions.

Let us illustrate the use of the debug capability with an example. To cause a break at a particular instruction, the user loads the starting address of that instruction into one of the breakpoint address registers, DR0 to DR3. The

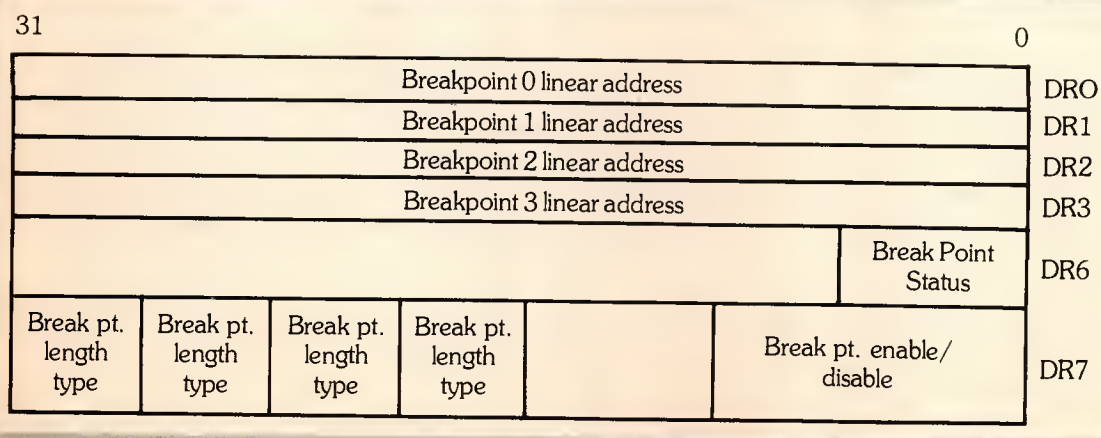
Table 3.
Data types supported by the 80386 instruction set.
(Floating point is available when numeric coprocessor is added.)

Operation	Data type: Ordinal	Integer	BCD	Floating point	String	Bit string
Move to/from memory, convert precision	X	X	X	X	X	X
Arithmetics add, subtract, multiply, divide, negate	X	X	X	X		
Logicals AND, OR, XOR, shift	X	X				
Compare	X	X	X	X	X	X
Transcendentals				X		

corresponding enable bit for the selected register must be set, and the type and length field must be set to instruction break (length = one byte). When the CPU is certain it is about to execute that instruction, it completes the execution of the current instruction, and a debug exception is generated. Note that if a successful branch or transfer of control precedes the intended breakpoint instruction, the break does not occur. An instruction break occurs before the instruction causing it is executed, whereas a data reference break occurs after the instruction causing it is executed.

The single-stepping-by-instruction feature forces an exception after each instruction execution. It can be used for system monitoring on an instruction-by-instruction basis. The one-byte trap instruction causes a software trap when executed and is useful for debugging exception-handling code.

80386 system debug registers.



notation. This allows a common set of instructions for addition and subtraction. For example,

```
SUB   ESP, 5
```

subtracts five from ESP whether ESP stores an integer or an ordinal. The settings of the overflow, sign, zero, and carry flags allow a program to determine whether a signed or an unsigned overflow has occurred. However, special instructions are provided for determining overflow in multiply and divide operations involving integers, since an integer multiply has its own rules for overflow and an integer divide produces its own unique bit patterns.

Pointers. A pointer is a memory address. There are two types of pointers in the 80386: near pointers and far pointers. A near pointer is another term for an effective address. A far pointer has two components: a word-sized selector and a d-word-sized effective address. The selector names the logical address space in which the effective address resides. This ability to define logical address spaces gives a user greater flexibility in structuring memory. (This is discussed in greater detail below.) To retain compatibility with 16-bit members of the iAPX 86 processor family, the 80386 also supports pointers having word-sized selectors and word-sized effective addresses.

Bit fields. The 80386 can do fetches from, or perform stores into, contiguous bit sequences of up to 31 bits each, where such bit fields themselves reside in a bit string of up to four gigabits. Single-bit values can also be tested and modified. Furthermore, bit fields can be scanned for the first set bit in either a forward or a reverse direction. This feature can be used to implement the *set* type of Pascal. For example, if *col* is an object of the type *set of color*, then the Pascal fragment

while c **in** col **do**

can be translated into

```
BSF   EAX, col   ; Find first set element. Store in EAX
JZ    loopExit   ; Exit if none left
```

Floating-point operations. By adding the 80287 floating-point coprocessor, or the higher-performance 80387, the user can extend the 80386 instruction set to support 32-bit, 64-bit, and 80-bit IEEE-standard floating-point arithmetic directly. These coprocessors provide the accuracy and performance demanded by numerically intensive applications such as robotics and graphics.

Multiprecision operations. The 80386 provides limited support for 64-bit operands. Integer and ordinal multiply and divide operations may have 64-bit products and dividends; the multiplier, multiplicand, divisor, quotient, and remainder are limited to d-word quantities. Multiprecision add and subtract operations can be easily synthesized with the add-with-carry (ADC) and subtract-with-borrow (SBB) operations.

Besides multiprecision arithmetic operations, the 80386 provides double-width shift instructions that accept a 64-bit input and generate a 32-bit output. These instructions can be viewed as generalizations of the normal logical shifts, except that the value shifted in is not zeroes but is specified by the contents of another 32-bit operand. Double-width shifts are especially useful for buffering intermediate data when performing operations on unaligned bit strings. A barrel shifter within the 80386 makes the execution times of these instructions independent of the size of the shift—any register-based, double-shift operation can be done in three clock periods.

Logical addresses. Thus far we have discussed memory addresses only in the context of effective addresses. We shall now investigate the logical address spaces provided by the 80386 to allow convenient memory partitioning. Each logical address space is named by means of a word-sized selector. All memory addresses have two components: the selector that names the logical address space (or segment) and an effective address (or offset) that indexes into the named logical address space. The full selector:offset form of address is the far pointer mentioned previously. The selector is not usually directly specified in an instruction's operand field; it is instead stored in a segment register.

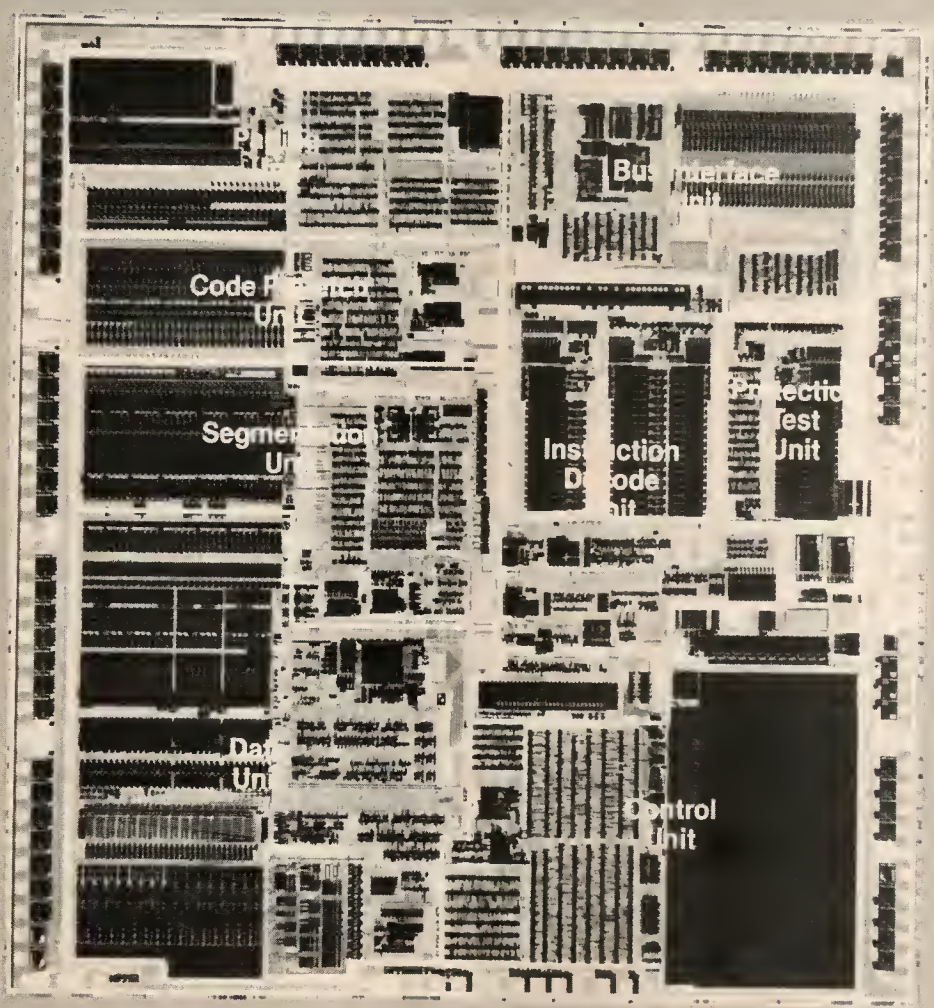
There are six segment registers named CS, DS, ES, FS, GS, and SS, as shown in Figure 4. Segment registers are not usually encoded in instructions; the segment register to be used is instead implied in the operand type. For example, code is fetched from the logical address space named by the selector in CS, at the offset specified by the instruction pointer EIP. Similarly, the stack is located in the logical address space named by the selector in SS, with the top-of-stack at offset ESP.

For memory operands not residing in the stack, the implied segment is usually DS. If a memory operand's default segment register is not desired, it

The 80386 physical implementation

The 80386 measures 390 mils on a side and is implemented in CMOS with Intel's CHMOS-III process, which provides 1.5-micrometer geometries and two layers of metallization. This allows 16-MHz operation with low power consumption. The chip contains over 275,000 transistors and is housed in a 132-pin pin grid array to simplify external interfacing and improve reliability.

To meet the high performance objectives of the 80386 architecture, the chip's designers organized the CPU into eight pipelined logical units and provided for a high degree of execution overlap among them. The units, which are shown in the die photo are the bus interface unit, the instruction (code) prefetch unit, the instruction decode unit, the segmentation unit, the paging unit, the protection test unit, the control unit, and the data unit. The last three comprise the execution section of the CPU, which consists of a microengine, a register file, an ALU, a barrel shifter, and miscellaneous control logic. On-chip memory management is implemented by the protection test, segmentation, and paging units.



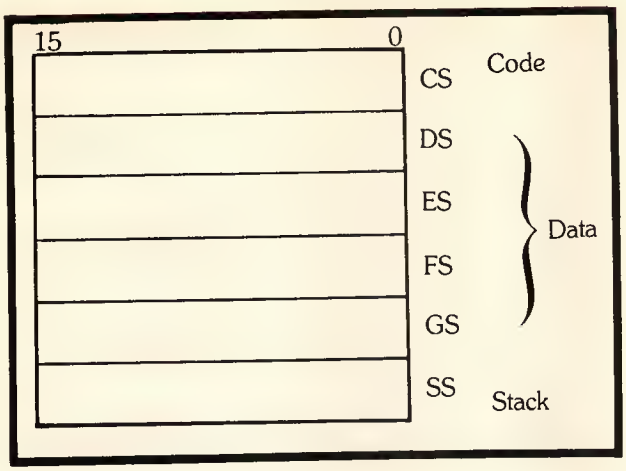


Figure 4. Segment registers.

can be overridden by placing one of six unique prefix bytes just before the instruction at which the override is to take effect. This allows rapid switching between different address spaces without requiring that a segment register be loaded with the correct selector value every time. It should be noted that segment descriptors are cached on the chip from their respective descriptor tables to allow rapid address translation and protection checking. Along with allowing the creation of address spaces, selectors form the basis for the 80386 memory management and protection scheme.

OS architecture—the memory management and protection model

Many computational environments require memory to be protected from unauthorized access. Furthermore, the allocation and deallocation of memory according to a process's needs must be managed. The 80386 provides a comprehensive set of mechanisms for supporting these requirements. The overall memory address generation structure is shown in Figure 5.

The logical address's selector and offset components specified by the instruction are mapped into a *linear address* via segment tables. Figure 6 shows how the selector specifies a segment descriptor. A segment descriptor is an eight-byte record. The main information it contains is the linear address and size of the base of the segment, and the type of reference that is allowed to be made into the segment. (Segment access rights are discussed below.) The linear address is constructed from a logical address by adding the offset to the linear base address. If this address exceeds the bounds of the segment (as specified by its size), a segmentation exception is signaled. An instruction encountering such an exception is fully restartable. Note that there are two segment tables of 8192 entries each (see Figure 6 again). This permits a total of 16,383 logical address spaces,* or roughly 14 bits of addressability. Since the offset furnishes 32 bits of addressability, the total logical address space of the 80386 provides 2^{46} bits of addressability.

The linear address is passed through a two-level page map table to generate a physical address. The physical address is the address delivered to the microprocessor's external bus for a memory access.

If a particular environment requires only one logical address space, the selector mechanism can be easily bypassed. This is done by defining one large logical address space spanning the entire linear address space and loading the corresponding selector into all segment registers. The difference between a logical address and a linear address then disappears, since the effective address component of the logical address matches the linear address. Similarly, if a linear address to a physical address mechanism is not required, paging can be disabled by resetting a bit in a control register. When paging is disabled, the linear address bypasses the page table look-up and appears directly as the physical address.

Since both translation steps are optional, the 80386 can allow the user to choose from one of four distinct views of memory:

- Unsegmented unpaged memory. Here both translation steps are bypassed, thereby making the effective address the same as the physical address. This is useful, for example, in a low-complexity, high-performance controller application, which requires a simple view of memory.
- Unsegmented paged memory. Here memory is viewed as a paged linear address space. Protection and management of memory is done via paging. This view is favored by some operating systems—e.g., Berkeley UNIX.
- Segmented unpaged memory. Here memory is viewed as a collection of logical address spaces. The advantage of this view over a paged approach is that

*Entry zero of the global descriptor table is special-cased to indicate a *null selector*. Hence, only a total of 16,383 descriptors is supported.

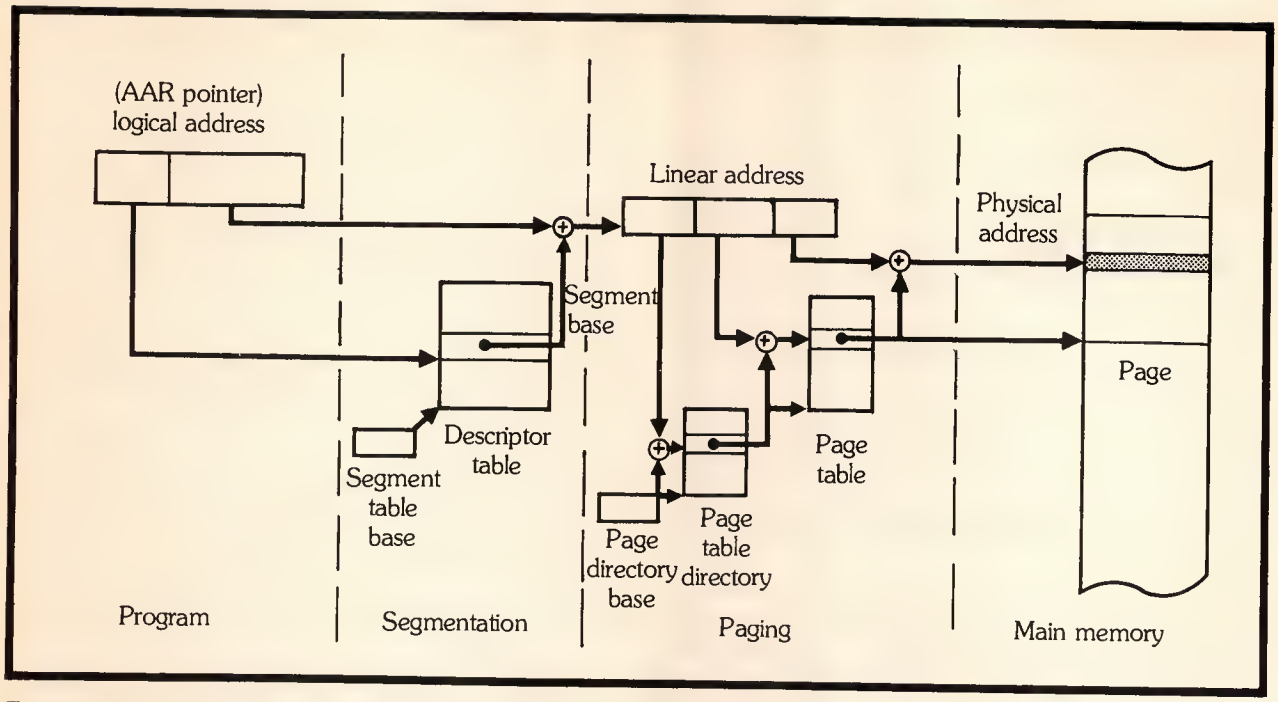


Figure 5. Memory address translation mechanism.

it affords protection down to the level of a single byte. Furthermore, unlike paging, it guarantees that when a linear address is generated the translation information is on-chip. Hence, segmented unpaged memory results in predictable access times.

• Segmented paged memory. This is the most comprehensive view of memory supported by the 80386. It uses segmentation to define logical memory partitions and paging to manage the allocation of memory within the partitions. Operating systems such as UNIX System V favor this view.

Segmentation mechanism. Memory protection is enforced in the 80386 through a concept called a *privilege level.* At any instant the processor is in one of four privilege levels. The current privilege level, or CPL, is stored as a number in the range 0 through 3, with level 0 being the most privileged level and level 3 the least. Furthermore, every logical address space (segment) has a privilege-level attribute associated with it called the descriptor privilege level, or DPL. The DPL is encoded in a field in the descriptor for a segment. Data access to a logical address space by a process operating at a given CPL is disallowed if the CPL is less privileged than the DPL of the logical address space. The CPL itself can change whenever CS is loaded with a new segment. Thus, the CPL is usually the DPL of the current code segment.

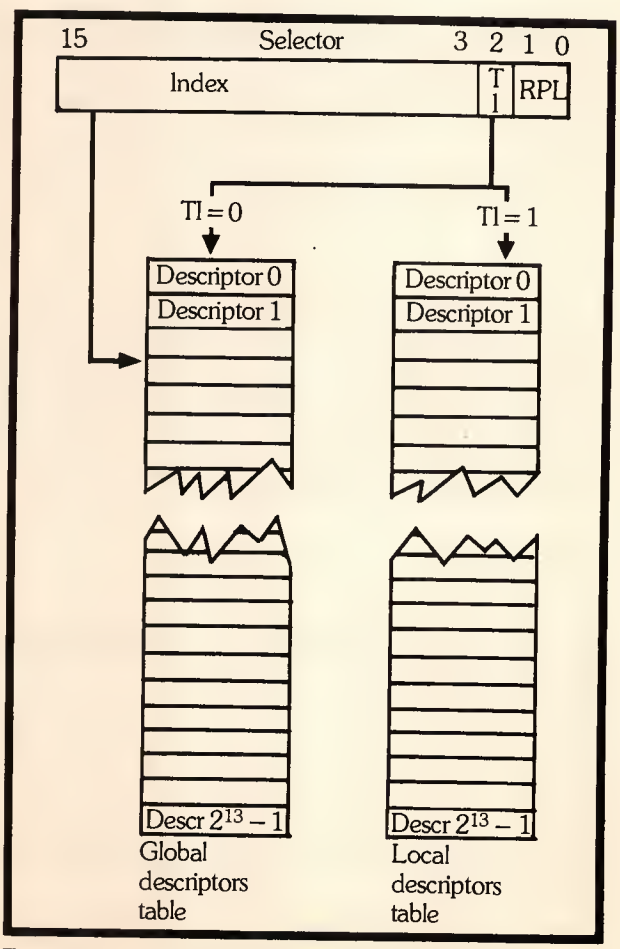


Figure 6. How a selector names a descriptor.

In addition to the DPL, a logical address space possesses an access attribute. This attribute is also stored in the descriptor for a segment. For address spaces containing data, it specifies whether read/write or read-only accesses are permitted. For address spaces containing code, it specifies whether read/execute or execute-only accesses are permitted. An access to a logical address space is allowed only if the access request passes a privilege level check and an access type check. The access attribute, privilege level, base address, and segment limit are contained in a segment descriptor. Each descriptor is an entry in either the global descriptor table (GDT) or the local descriptor table (LDT). The GDT specifies logical address spaces shared by all tasks, whereas the LDT specifies logical address spaces specific to a single task.

To support memory management, all descriptors have a *present bit* and an *accessed bit*. An access to a logical address space whose present bit is clear causes a fault, which is signaled via an interrupt. The fault handler can bring the logical address space's contents into the linear address space, set the linear address space's base and limit in the descriptor, and set the present bit. The faulting instruction can then be restarted. This implements a demand-swapping scheme for logical address spaces. The accessed bit is provided to help with the replacement policy implementation. This bit is set whenever a selector defining a logical address space is loaded into a segment register, thereby indicating which logical address spaces have been used recently.

Paging mechanism. To complement logical-to-linear address translation, the linear address is translated via paging into a physical address, as shown in Figure 5. The paging scheme involves a standard, two-level, table look-up process. The linear address to be translated is divided into three fields: a directory table index, a page table index, and a byte index. The directory table index is a 10-bit field that selects one of 1024 page tables. This page table is in turn indexed by the 10-bit page table index, which selects one out of 1024 pages. This page is a 4096-byte block which is indexed by the 12-bit byte index. The byte thus addressed specifies the low-order byte of the operand addressed. The entries in the directory and page tables are d-word quantities that store the physical base addresses of page tables (for directory tables) or of pages (for page tables). These entries also provide the traditional dirty, accessed, and present bits to allow for the implementation of replacement policies in demand-paged systems. The layout of each entry is shown in Figure 7.

The paging mechanism also provides protection at the page level. This is done via the user/supervisor and read/write bits in the directory and page table entries. A process is considered to be executing in user mode if CPL is set to 3; it is considered to be executing in supervisor mode if CPL is 2, 1, or 0. Therefore a user process is allowed to read a page only if both the directory entry and page table entry for it have the user/supervisor bit set. Similarly, a user write is allowed only if both entries have both the user/supervisor bit and the read/write bit set. A supervisor process is allowed to read or write all pages without restraint. Any violation of the page protection rules causes a paging exception. An instruction encountering such an exception is fully restartable. This permits the construction of demand-paged systems and, it should be noted, allows the use of the copy-on-write trick in implementing UNIX's *fork* primitive.

Task switching support. In addition to the memory management and protection facilities described above, the 80386 assists the operating system by providing hardware to implement task switching. This

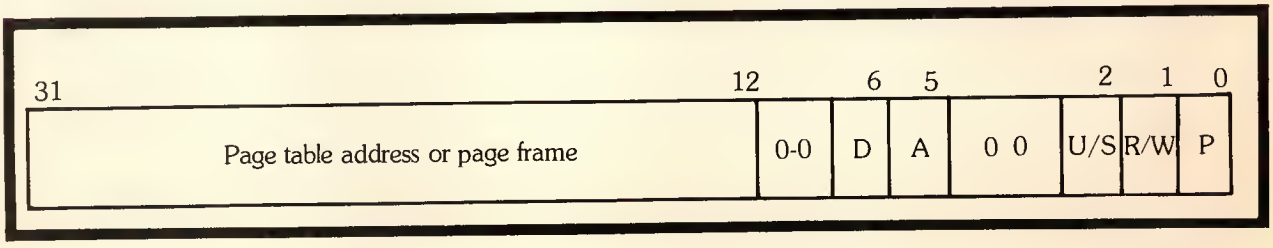


Figure 7. Directory table and page table entries.

hardware allows fast and efficient task switching in multitasking applications.

In task switching, the state of a task is stored in a data structure called a task state segment, or TSS. The fields of the TSS store images of the general registers, flags, instruction pointer, and segment registers, and an image of a pointer to the page directory table base and the local descriptor table. The TSS is itself described by a specially tagged descriptor residing in the global descriptor table. For a task switch to be performed, a call or jump is made via a selector that names such a descriptor. The processor recognizes the TSS and does a task switch by saving the current machine state in the currently active TSS and loading the state in the target TSS.

Compatibility with the 80286. Full compatibility with the 80286—to protect investments in operating systems and applications software—was clearly one of the key objectives of the 80386 architecture. Such compatibility was achieved by making the 80386 instruction set object code compatible with the 80286 instruction set; all architectural extensions to the 80286 instruction set, data types, and addressing modes adhere to strict compatibility rules. Furthermore, as the discussion earlier in this article showed, the basic 80286 memory management model was maintained and extended by the 80386.

Compatibility with the 8086. The 80386 is compatible with the 8086 in the same way that the 80286 is compatible with the 8086. The 80386 is similar to the 80286 in that it too powers up in 8086-compatible mode (also called real mode). Real mode is useful for initialization and for configuration of the processor data structures needed to run in native 80386 protected mode. The recommended method for running 8086 code is called virtual 86 mode.

To allow 8086 code to run harmoniously with its native code, the 80386 employs the notion of a virtual 8086, or VM86, task. Except for I/O and interrupt-related instructions, a VM86 task executes code using 8086 semantics. Memory addresses generated are treated as linear addresses and are subject to translation via paging. Mapping each VM86 task's linear address space to a different physical address virtualizes 8086 memory. However, operations that use global resources, such as I/O and interrupt operations, also need to be virtualized. This is done by trapping the instructions for such operations and allowing the virtual machine monitor (executing as a native 80386 program) to emulate them. VM86 mode is a key feature of the 80386—it allows the large body of 8086 software to run concurrently with native code in high-performance environments, and thereby allows 86, 286, and 386 tasks to run simultaneously.

80386 implementation

To meet the 80386's performance objectives, the chip's designers gave careful consideration to the internal implementation of the architecture—i.e., the microarchitecture. Performance enhancements in advanced microprocessor CPU designs can be obtained through

- advances in technology and circuit design,
- higher clock rates,
- wider data paths,
- advances in system and bus architecture, and
- advances in microarchitecture.

The first four of these were discussed earlier along with their effects on performance. The following sections describe the 80386 microarchitecture, focusing on performance-enhancing features such as pipelining and parallelism.

The 80386 is organized as eight logical units, with each unit assigned a task or step in the fetching and execution of each instruction. This arrangement allows as much parallel execution of the instruction stream as possible. The units are pipelined and, for the most part, operate autonomously. The units and their interconnections are shown in Figure 8 in the functional block diagram of the 80386.

The eight units are the bus interface unit, the prefetch unit, the instruction decode unit, the control unit, the data unit, the protection test unit, the segmentation unit, and the paging unit. The control, data, and protection test units comprise the execution section of the CPU.

The bus interface unit interfaces the CPU to the external system bus and controls all address, data, and control signals to and from the CPU. The prefetch unit is responsible for fetching instructions from memory. It uses an advance-instruction-fetch pointer to prefetch code from memory and store it in a temporary code queue. This queue also acts as a buffer between the prefetch unit and the instruction decode unit. Since addresses generated by the prefetch unit are linear, they must be translated to physical addresses by the paging unit before the prefetch bus cycle request can be sent to the bus interface unit.

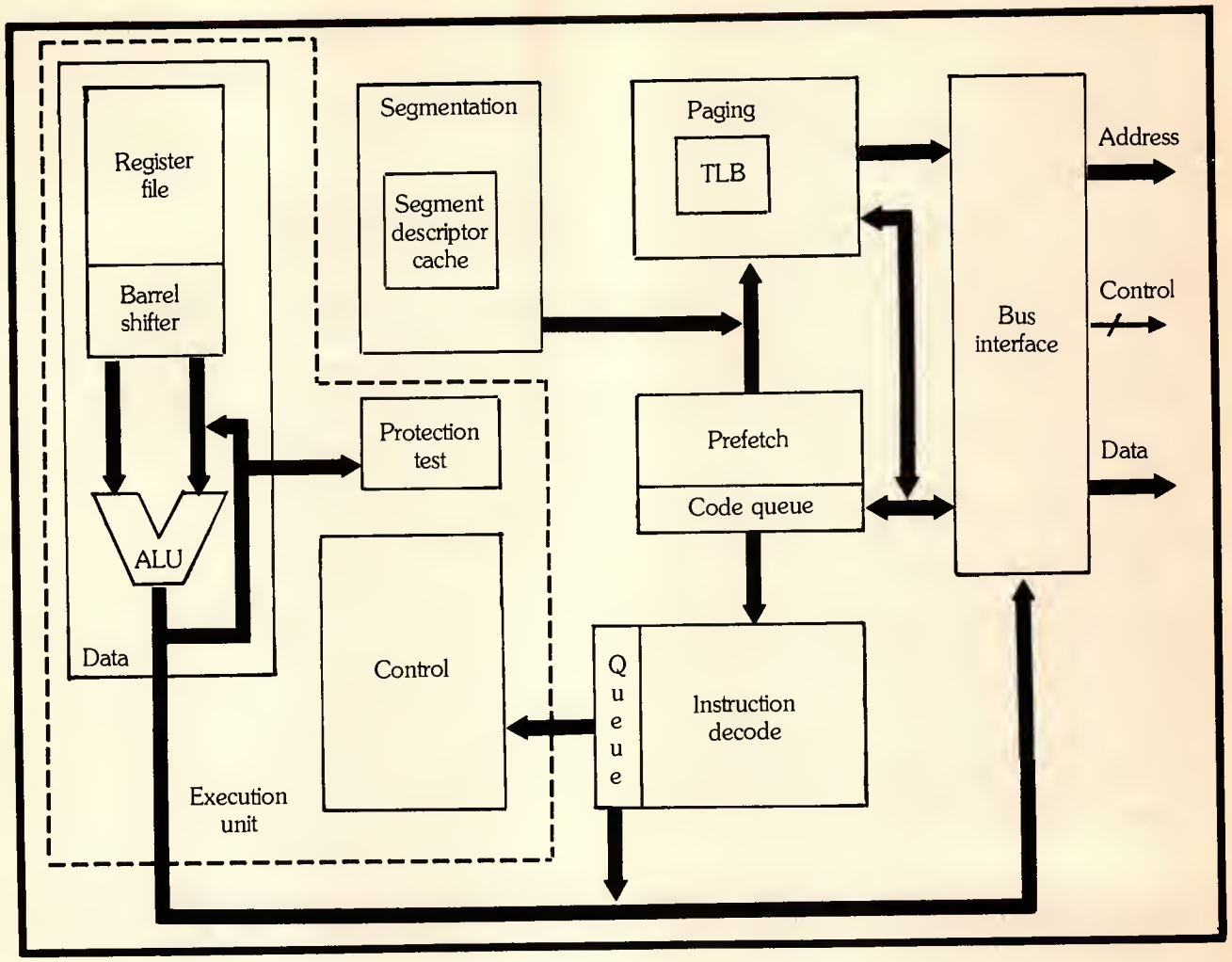


Figure 8. Block diagram of 80386.

The instruction decode unit prepares and decodes instructions for immediate execution by the execution unit. It does this by fetching bytes of code from the prefetcher's code queue, transforming them into a fully decoded instruction, and then storing that instruction in a three-level decoded instruction queue. The execution unit then operates on the decoded instruction, performing the steps needed to execute it.

Instructions requiring memory references send their requests to the segmentation unit for logical address computation and translation and segment protection violation checking. The segmentation unit produces a translated linear address which the paging unit then translates into a physical address. The paging unit also checks for paging violations before it sends a bus request and the address to the bus interface unit and external bus.

Pipelining and parallelism. Advanced microprocessors are normally pipelined by overlapping the fetching, decoding, and execution of instructions. In the 80386 microarchitecture, however, the operations of all eight of the logical units are overlapped. This allows the parallel and autonomous operation of the units. They can simultaneously operate on different instructions, thereby significantly boosting the overall instruction processing rate of the CPU. For example, while the bus interface unit is completing a data write cycle for one instruction, the instruction unit can be decoding another, and the execution unit processing a third.

The sections below describe each of the logical units of the 80386 and discuss how each was designed to maximize the benefits of pipelining.

Bus interface unit. The bus interface unit provides a high-speed interface between the CPU and the system. Its function is to efficiently meet the CPU's requirements for external bus transfers during code fetches, data fetches, paging unit requests, and segmentation unit requests. To accomplish this, it has been designed to accept and prioritize multiple internal bus requests so that it can make maximum use of the available bus bandwidth in servicing those requests. This activity is overlapped with any current bus transaction. As mentioned earlier, the 80386 bus uses only two clocks per cycle; if the pipelined mode is used, the 80386 bus is capable of starting the next address of a new bus cycle before the completion of a current bus transaction.

Prefetch unit. This unit is responsible for prefetching instructions from memory. It stores the aligned code in its code queue for efficient decoding by the instruction unit. It maintains a linear address pointer and a segment prefetch limit that are initially obtained from the segmentation unit to be used as a prefetch instruction pointer and for checking segment limit violations, respectively. The prefetcher attempts to keep its code queue filled with valid bytes of code by sending prefetch bus cycle requests to the bus interface unit through the paging unit. Prefetch bus cycle requests are made whenever the prefetch code queue is partially empty or after the occurrence of a control transfer. The prefetcher's bus cycle requests are assigned a lower priority than execution-related, operand fetch/store bus cycle requests and page-miss processing and segmentation-specific bus cycle requests. At zero wait states, there is no interference between prefetch and data bus cycles. Idle cycles are used to prefetch code from memory and keep the code queue filled.

Instruction decode unit. This unit decodes and prepares instructions for processing by the execution unit. Whenever the instruction unit's own queue or pipe is partially empty, it fetches bytes of code from the prefetcher code queue, decodes and prepares them, and stores the result in its own three-word-deep queue. The decoded instruction queue words are very wide; they contain all the instruction fields the execution unit needs to immediately execute the instruction without further decoding. The combined prefetch unit/instruction unit pipe operates on a two-clock cycle. The instruction unit, however, can decode at only one clock per opcode byte.

Execution unit. The next logical unit in the pipe is the execution unit. As mentioned previously, it is composed of the control, data processing, and protection test units. Its responsibility is to execute the instruction given to it. It does so by using its own resources as well as by communicating control and sequencing information to other logical units needed to complete the execution of the instruction. The fully decoded instruction is popped out of the instruction queue, and the execution unit uses its various fields, such as microcode starting addresses, operand references, data types, and ALU operators, to execute it.

The control section consists of a microcode-driven engine that has special-purpose hardware for decoding, assisting, and speeding up microcycle execution. The data processing section—or data path—contains all data registers, an ALU, a barrel shifter, multiply/divide hardware, and special control logic; it performs the data operations selected by the control section. The protection test section performs all static segmentation-related violation checks under microcode control.

The microengine has a two-clock execution latency, but by overlapping microinstruction fetching and execution and by using the delayed microjump technique, it provides an execution rate of only one clock per microcycle (62.5 ns at 16 MHz). To enhance the effective instruction processing rate of the execution unit further, parallel or overlapped execution of instructions is employed. Since memory reference instructions, including stack push and pop instructions (heavily utilized in procedure calls), constitute a large portion of the instruction mix in a typical program, a special technique is used to reduce the number of clocks needed to execute such instructions. The method used partially overlaps the execution of every memory reference instruction, including stack push and pop instructions, with the execution of the preceding instruction. This parallel execution of two instructions enhances the instruction processing rate of the CPU—with a typical mix of instructions, it yields a nine-percent improvement in performance.

The implementation of the execution unit required the addition of a 32-bit internal bus and special control logic to ensure the correct completion of the current instruction, prevent the use of stale register values, and provide the control needed to handle the simultaneous execution of two instructions.

Segmentation unit. This unit performs effective address computation upon request of the execution unit. It does this logical-to-linear address translation at the same time it does bus cycle segmentation violation checks. (Static violation checks—e.g., of seg-

ment descriptors—are performed by the protection test unit and are not part of the bus cycle activity.) The translated linear address is then sent to the paging unit along with bus cycle transaction information. The paging unit then becomes responsible for requesting bus service from the bus interface unit.

Paging unit. Linear addresses generated by the segmentation or code prefetch units are passed on to the paging unit, where they are translated into physical addresses. As explained in the section on architecture above, paging translation is implemented through a two-level page relocation mechanism. To improve performance, the paging unit uses a 32-entry translation look-aside buffer (TLB) to perform the translation. Page table and page frame entries are cached into the TLB. This results in a translation time of one half of a clock period—much faster than if memory-based page tables were referenced instead of the TLB. The combined logical-to-physical address translation pipe, which includes effective address formation, segmentation, and paging relocation, requires only two clocks of processing. No additional clocks are needed for paging since TLB look-up and translation are performed in the same clock (second phase) as the linear address calculation.

Integrated segmentation and paging. The 80386 implements a segmentation-plus-demand-paging memory management function for virtual memory translation and protection violation checking. To enhance system performance, memory management functions are integrated into the chip. All virtual-to-physical address translation, segmentation, and paging violation checking are performed by the on-chip memory management unit (MMU), which is imple-

80386 bus operation

The 80386 external interface (Figure B) connects the chip to memory and peripherals over a high-performance bus. This interface provides separate address and data pins to support efficient high-speed bus transfers and bus cycle pipelining. It uses its other pins to maintain simple and efficient interfaces—pins are dedicated to cy-

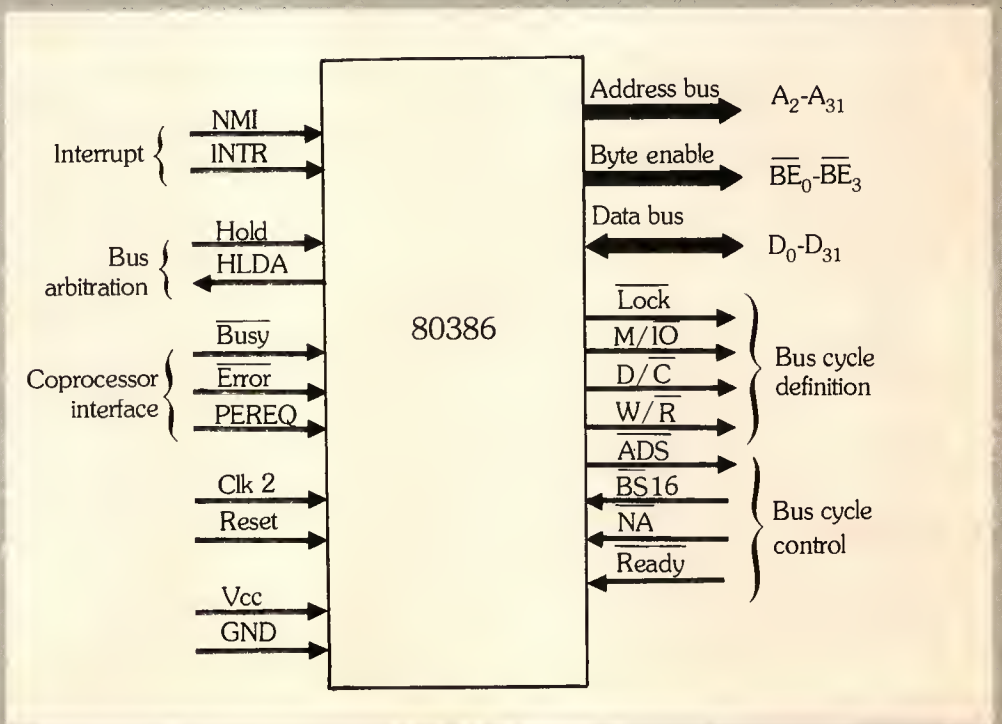


Figure B. iAPX 80386 external interface.

mented by the segmentation, protection test, and paging units. All memory management protection and translation descriptors are cached into the on-chip MMU. Segment descriptors are cached into segment descriptor caches in the segmentation unit, while page descriptors are cached into the TLB. Virtually all protection and translation activities utilize the on-chip descriptors.

There are several advantages to integrating memory management and the CPU on the same chip:

• The implementation can take advantage of pipelining and parallel execution by overlapping the steps from effective address formation to linear address and physical address generation. Under such an arrangement the next memory address will already be known while the current transaction is still on the bus. The logical-to-physical address translation in many cases will overlap other bus cycles and will be completed by the time the current bus cycle is finished. Pipelining to an off-chip MMU would clearly be more difficult.

• No additional clocks need to be added to the bus cycle to implement the MMU translation, since the memory management functions are internal and not part of the external bus cycle.

• System complexity and cost are reduced, and board design is simplified, since no off-chip memory management components are needed.

Special-purpose hardware. To enhance performance, the 80386 implements several important protocle definition, cycle control, interrupts, bus arbitration, and coprocessor support.

Some typical 80386 bus cycles are shown in Figure C. With zero wait states, each bus cycle consists of only two clocks; at 16-MHz operation, the 32-bit bus can sustain a 32-megabyte-per-second transfer rate. When the bus operates in pipelined mode, the address of the next bus cycle is sent on pins A31-A2 before the current cycle is completed. This mode, which is dynamically selectable from the interface, can be used to implement a fast interleaved memory subsystem with inexpensive dynamic RAMs. In such a system, the current bus transaction uses one bank of memory while the new address is used to access another. The pipelined mode can also be used to overlap the current bus cycle with address decode delays of the next cycle, making available longer system access times. Slower memory or I/O systems use the $\overline{\text{READY}}$ pin to extend the current bus cycle as needed. Dynamic bus sizing allows software-transparent interfaces to 16-and 32-bit ports.

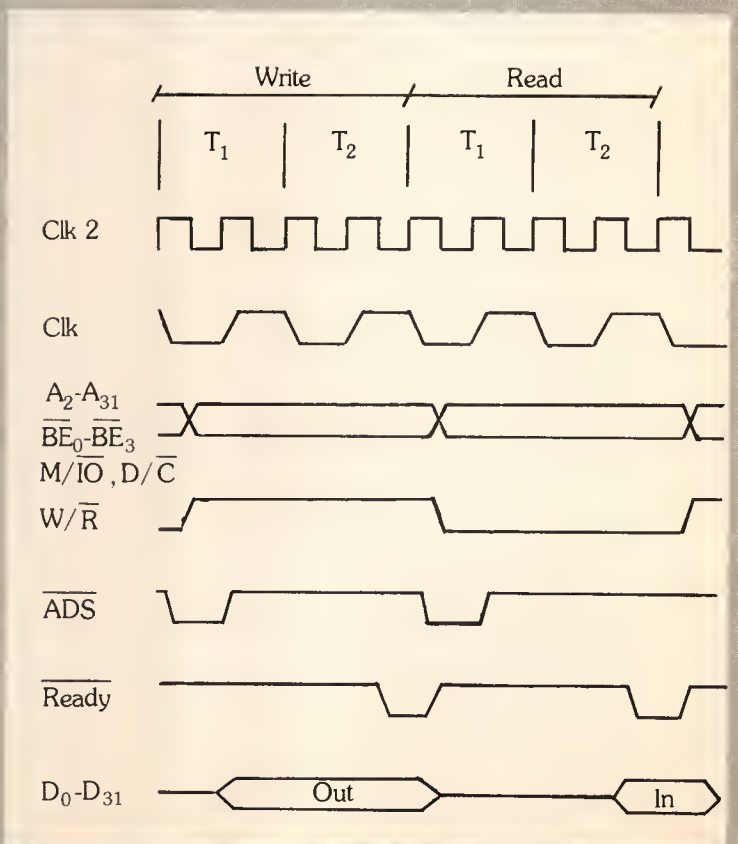


Figure C. 80386 bus operation.

cessing functions with special-purpose hardware. These special blocks include the TLB for paging translation, a 64-bit barrel shifter, multiply/divide hardware, multiple ALUs and adders, and miscellaneous random-logic control function implementations that take the place of PLA or microcode approaches that are too slow.

Translation look-aside buffer. As explained above, the time required to perform the two-step table look-up paging translation function is significantly reduced by the use of a TLB. With 32 entries in its cache, the TLB enjoys a relatively high hit ratio of 98 percent. A TLB miss occurs when an entry is not in the cache and references to memory-based page

Design for testability

To guarantee high quality in a complex VLSI microprocessor, the chip development team must

- use good, conservative design techniques,
- employ rigorous design verification at all levels,
- carefully choose the criteria against which manufacturing, testing, and quality assurance will be measured, and
- ensure the testability of the device.

Intel's goal is to manufacture the highest-quality silicon as inexpensively as possible. Adequate device testing helps achieve that goal. However, the testing of complex VLSI circuits is difficult. Here we explore the features that have been designed into the 80386 to make testing easier and ensure its adequacy.

A VLSI microprocessor is difficult to test for two reasons:

- It contains a large number of transistors but has a small number of external pins. The external pin count is too small to allow for a convenient and thorough interface to all the internal blocks of the microprocessor.
- It exhibits a fairly large number of states. It is virtually impossible to force the microprocessor to sequence through all possible states under all possible conditions.

These limitations make it very difficult to control and observe a VLSI microprocessor that is being tested.

To alleviate these problems, the 80386's designers incorporated test circuits into the device to ensure its testability. They designed these circuits to support proven test techniques. The test capability thus added facilitates component testing in the lab and in production environments and simplifies board and system testing in end user applications. Test capabilities built into the 80386 include

- self-test of all large PLAs,
- self-test of the control ROM and microengine, and
- signature analysis.

Special microcode and instructions are also provided to facilitate testing, and there is special test circuitry for the translation look-aside buffer (TLB).

PLA testing. In the 80386, self-test of large PLAs is needed because of the large number of possible input combinations. A maximal polynomial counter is used as a pseudorandom vector generator to drive the inputs to the PLA and to ensure sequencing through all possible combinations of the inputs. This is illustrated in the accompanying figure. As the inputs change in a pseudorandom fashion, the outputs from the PLA are fed into a linear feedback shift register with appropriate polynomial coefficients to accumulate a unique *signature* of the block. This signature is then read and analyzed by special software to determine correctness. This technique is referred to as *signature analysis*.

Signature analysis has been used for many years—primarily at the board and system level—to facilitate testing and fault diagnosing. It has the advantage of being able to reduce a large amount of data, uniquely encode it, and store it in a register (the linear feedback shift, or signature, register). It should be noted that the polynomials used to configure this register must be carefully chosen to provide maximum test coverage and error detection.

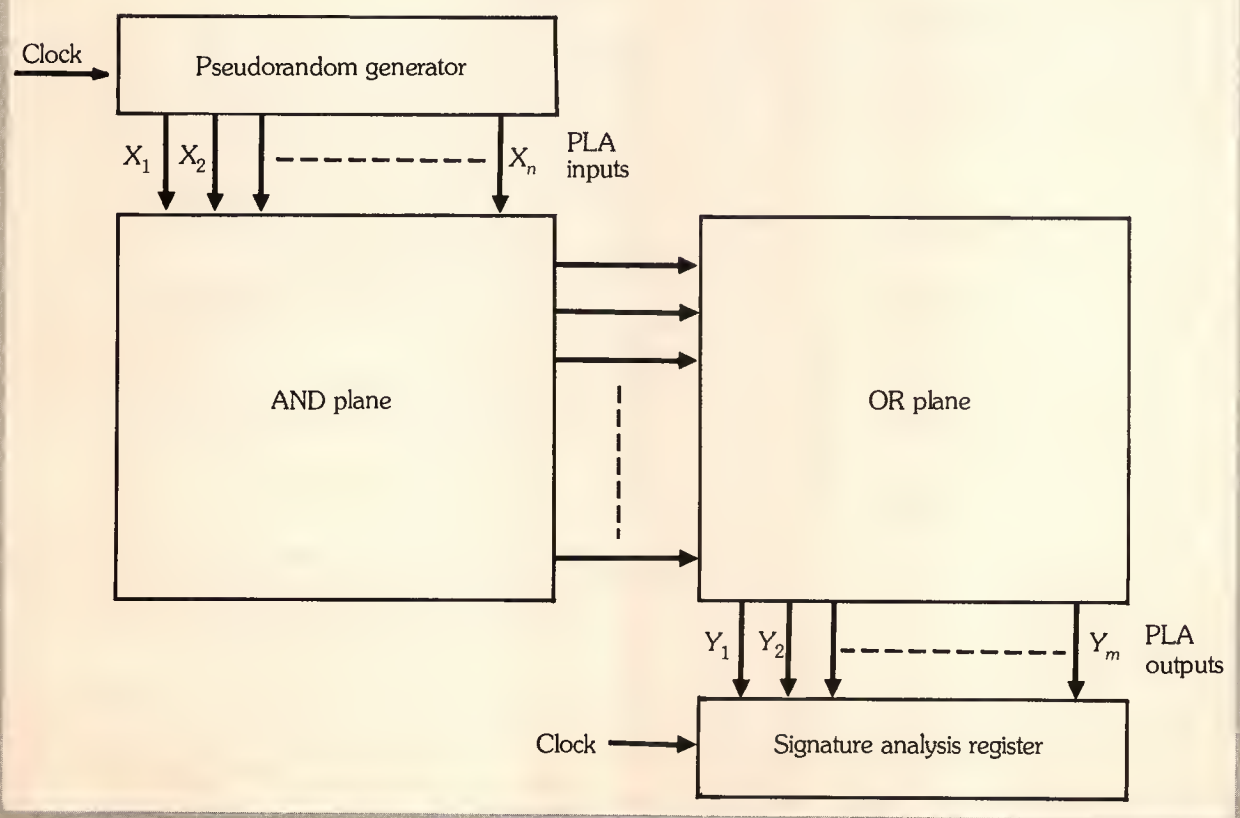
Control ROM testing. Self-test of the microcode ROM consists of sequencing through all addresses and word locations and, as in PLA testing, accumulating the result in a signature register. The accumulated microcode signature is then read by software. The microcode address counter logic is used to do the sequencing in the

tables are needed to complete the address translation. Even with a high hit ratio, the memory-referencing table look-up function is clearly time-consuming and should be designed to minimize the degradation of system performance. The 80386 paging unit incorporates special-purpose hardware to implement the table look-up functions instead of using the more economical but slower microcode approach. TLB miss processing by this special hardware consumes only nine clocks, whereas a microcode-driven look-up would consume three times as many clocks.

Barrel shifter. A 64-bit barrel shifter is provided to implement shift, rotate, and bit manipulation in-

ROM test sequence, to share hardware, and to provide address sequencing logic for the microengine self-test.

TLB test circuitry. Testing the TLB RAM and CAM (content-addressable memory) for all entries and with all data sensitivity patterns would be very cumbersome and time-consuming without special assistance. To simplify such testing and improve its coverage, the 80386's designers added special-purpose hardware—two registers, plus special instructions to control them—to allow for the direct writing of any data pattern into the RAM or CAM blocks of the TLB. This hardware can also force the comparison of any data pattern against existing entries in the TLB to test the TLB's matching capability.



PLA testability.

structions efficiently as well as to assist in multiply and other miscellaneous operations. It can shift any data type by any shift count in a single clock, and it is significantly faster than an ALU/microcode implementation.

Multiply/divide hardware. A one-bit-per-clock multiply/divide mechanism permits a 32-bit multiply or divide in 40 clocks, maximum. To speed multiply execution, additional special hardware is included to detect early completion of a multiply and correctly terminate the operation. Early completion of a multiply is detected when all significant multiplier bits have been exhausted and the final product can be obtained by an appropriate shift operation. Since many multiply operations do not require the full 32-bit multiply, the average number of clocks needed for a multiply is 20.

The 80386 represents the state of the art in 32-bit microprocessor architecture and performance. It maintains object code compatibility with existing members of the 86 family of microprocessors, thus protecting end users' investments in software. Its implementation exploits pipelining and parallel execution to provide high performance. These characteristics make it an excellent candidate for application in engineering workstations, office systems, and robotic and control systems.

Acknowledgments

The authors wish to acknowledge the superb efforts of the entire 80386 team, including the project managers, the chip architects group, the design engineers, and the mask designers.

Khaled A. El-Ayat is a project manager in the Intel High-Performance Microprocessor Operation. With Intel eight years, he has contributed to the definition, design, and development of the company's microprocessors. He designed the control structures of the 80386. His technical interests include microprocessor architectures and all aspects of VLSI design.

A member of the IEEE, El-Ayat holds a BSc from Cairo University, an MSc in electrical engineering and computer science from the University of Toronto, and a PhD in electrical engineering and computer science from the University of California, Santa Barbara. He is a part-time lecturer in microprocessors at the University of Santa Clara.

Rakesh K. Agarwal is a senior design engineer in the Intel High-Performance Microprocessor Operation. He contributed to the architectural specification, microcoding, and logic design of the 80386. His primary technical interest is in the definition of the interface needed between hardware and software to make the most optimal use of computer system resources. He is also interested in the design of CAD tools that can be used to assist the architecture-to-logic transformation.

Agarwal received the BSc in computer science from the University of British Columbia and the MSc in computer science from the University of Toronto.

Questions about this article can be directed to El-Ayat at Intel Corporation, Mail Stop SC4-59, 2625 Walsh Avenue, Santa Clara, CA 95051.

The Z80000 Microprocessor

David Phillips

Zilog, Inc.

Although the Z80000 is Zilog's first 32-bit machine, it is an example of the second generation of 32-bit microprocessors. It incorporates on a single monolithic chip all the major functional parts of a large computer architecture. These include memory management, multiprocessing and coprocessing support, cache memory, and extensive pipelining. The large register set and regular architecture, coupled with machine instructions that directly support high-level languages, provide a rich software development environment for large applications. Protection features insulate the operating system from corruption while providing means for communication with application tasks. Programmable bus interface control simplifies hardware design by providing for automatic wait-state generation, bus-speed scaling, and support of fast burst-mode memory transfers. Fast clock speeds combined with a 256-byte instruction and data cache and a six-stage pipeline provide performance levels that should range to five MIPS.

The applications for this high-performance processor will include multiuser office systems, graphic workstations, communications switching systems, and military systems. With the high level of integration provided by the Z80000, the microprocessor system designer should see his products take a quantum leap in performance.

Here, we will survey the architectural features of the Z80000 that directly support operating system and high-order language requirements, and we will describe the chip hardware responsible for the high performance level of the machine.

Operating system and high-level-language support

The register set of the Z80000 provides the programmer with a completely regular and highly flexible register model that can be mapped directly onto almost any operating system, high-order language, or application-specific requirement. The register set consists of sixteen 32-bit general-purpose registers that can hold addresses or data. Data may be accessed as bytes, words, or long words. Data are packed rather that telescoped into registers, providing maximum efficiency in register space usage. Data stored in the first four 32-bit long-word registers can be accessed as bytes, thus providing up to sixteen byte registers. Similarly, data stored in the first eight 32-bit registers can be accessed as words, providing up to sixteen word registers.

These attributes allow the programmer to model and expand the register set to fit his application. Since most tasks require a combination of byte, word, and long-word registers, the register set can be mapped onto the data model as required. For example, an application may require six byte registers, eleven word registers, and several 32-bit registers. As shown in Figure 1, the byte and word registers can be mapped onto the first seven long-word registers, leaving nine long-word registers for other functions. This model provides 26 working registers to the application programmer.

Push and pop instructions may use any of the registers other than the long-word register RR0 as a stack pointer. Call instructions, traps, and interrupts use the implicit stack pointer value contained in RR14 to store status information.

For operating system protection the Z80000 provides two modes of operation: system and normal. A set of privileged instructions is reserved for the system mode of operation, protecting the operating system and system hardware from the destructive actions of application programs. Instructions that perform I/O operations or affect internal configuration registers are examples of privileged instructions.

In addition to preventing application tasks from executing privileged instructions, the system mode of

0272-1732/85/1200-0023$01.00 © 1985 IEEE

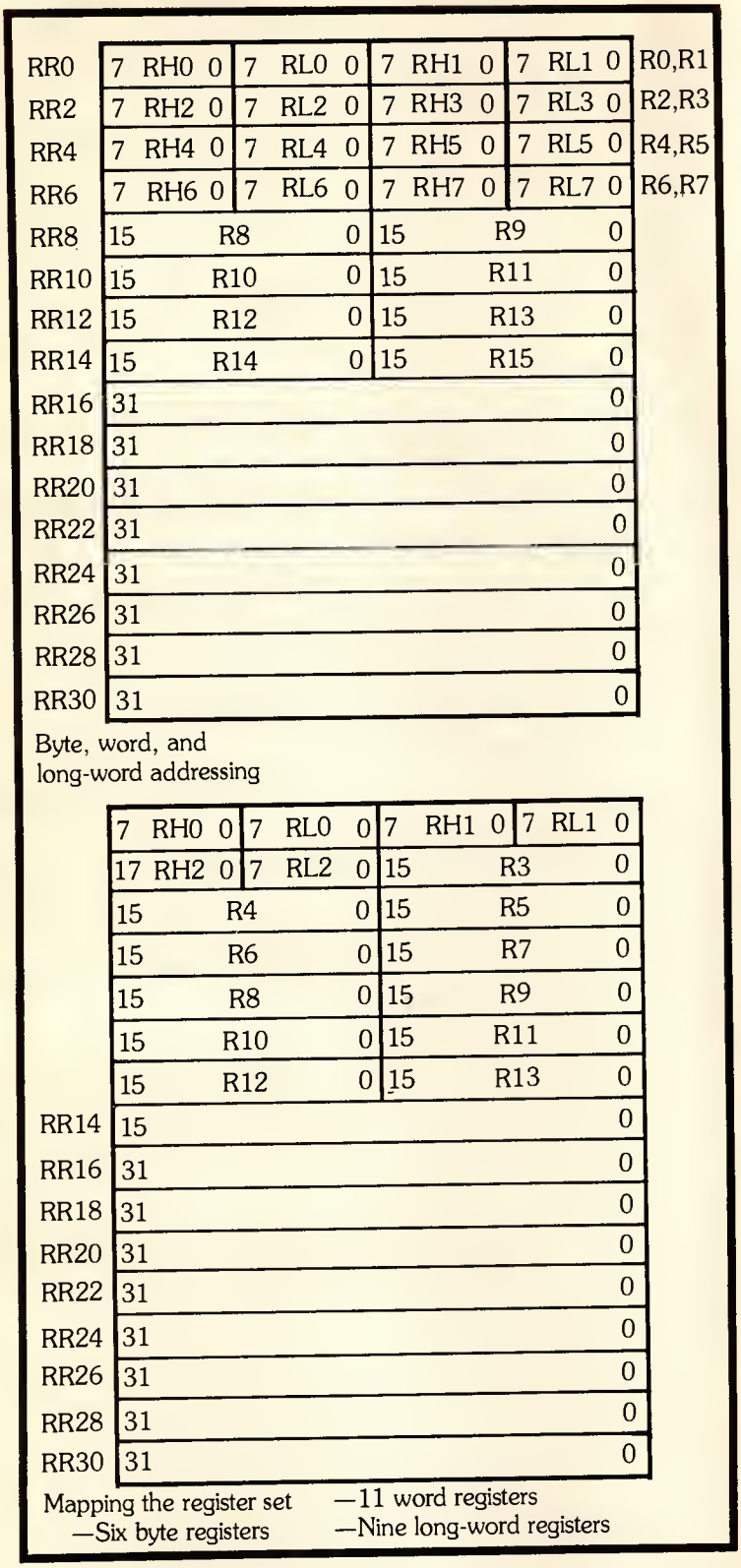


Figure 1. The sixteen 32-bit registers can hold addresses or data. Data can be accessed as bytes, words, or long words. The register model can be mapped to the requirements of a specific application.

operation provides hardware protection for the operating system. Dual stack pointers (RR14) are provided to ensure the integrity of the system stack. The processor automatically enters system mode when a trap or interrupt occurs. The system call instruction provides the means for application programs to request system resources.

An external status pin is provided that indicates whether the processor is in system or normal mode during a memory operation, allowing physical memory to be divided into separate system and normal address spaces. Privileged instructions are provided to allow the operating system to access normal memory space. This group of privileged instructions generates the external status for normal memory accesses. These instructions provide a convenient means for the operating system to retrieve block data from an application task's memory space. The system call instruction provides the means for an application task to request system resources. If parameters or data that must be passed are relatively compact, then they may be passed directly in registers.

Addressing modes and high-level instructions provide high-order language (HOL) support. The enter and exit instructions are designed specifically for subroutine calls and returns. The overall effect of the enter instruction is to perform all the overhead activity required to generate an activation record on the stack for a nested subroutine call. As shown in Figure 2, this instruction takes two operands. The "mask" operand specifies a subset of the register file to be pushed onto the stack. The "siz" operand allocates an area at the top of the stack for local storage of temporary variables used by the subroutine. The frame pointer, held in dedicated long-word register RR12, points to a location just below the local storage area which may be used to hold an exception handler address. Since the previous frame pointer value is pushed onto the stack, nested subroutine calls are supported.

The exit instruction unwinds the enter instruction. It pops local storage by replacing the stack pointer value with the frame pointer value. It then uses the pushed mask operand to restore the saved registers and reloads the old frame pointer value.

The check and index instructions are used to access arrays. The index instruction checks a subscript value against upper and lower bounds. If no violation is found, the array element address is calculated, taking into consideration a scaling factor. For multidimensional arrays, this instruction performs one step in the

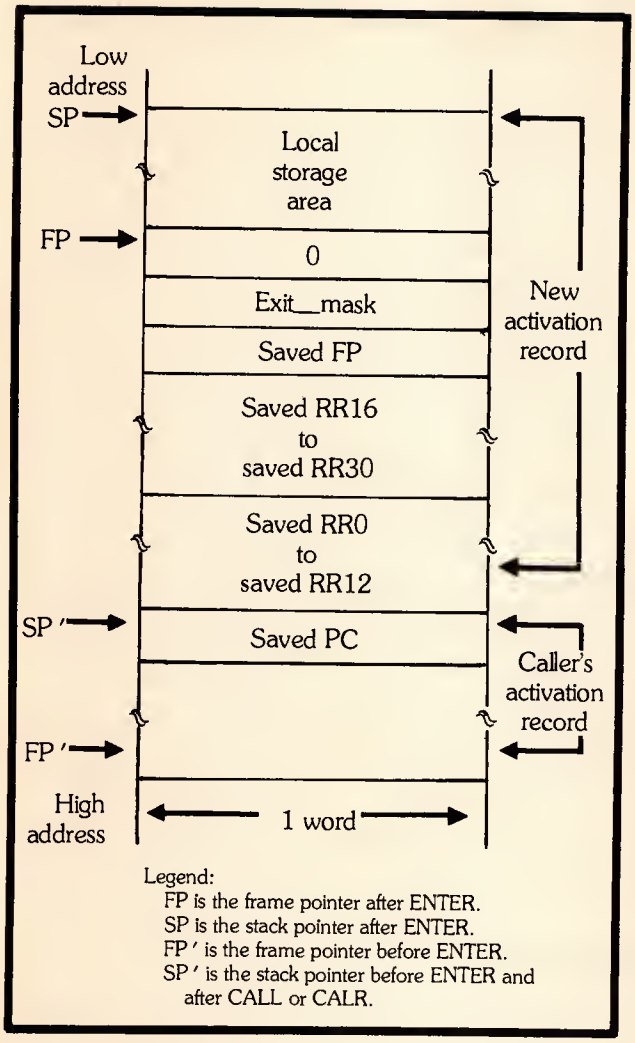


Figure 2. The enter instruction generates an activation record on the stack for a subroutine call. Storage for local variables is allocated, and the mask variable and the subset of the working registers that it specifies are pushed onto the stack. The exit instruction unwinds the enter instruction.

index calculation and may be reapplied as required. The check instruction performs only the subscript boundary check. With either instruction a boundary violation causes a system trap.

The bit-field insert and extract instructions provide a convenient means for storing and retrieving values in packed data types. For example, each element in an array of packed records representing employee information may contain a seven-bit binary field indicating job grade. The index and bit-field insert and extract instructions used in combination provide a simple means for reading or updating this field.

To support efficient binary encoding of machine instructions and consequently minimize the number of bytes required for instruction storage, the addressing modes of the Z80000 are restricted to the major ones needed for general operations and HOL support. However, the load address and load address relative instructions allow the programmer to construct the more exotic addressing modes that are occasionally needed.

The nine addressing modes of the Z80000 are register, immediate, indirect register, direct address, indexed, base address with displacement, base index with displacement, relative address, and relative indexed. While most of these addressing modes are straightforward in their meaning and application, the base index with displacement has special applicability to HOLs. As shown in Figure 3, the effective address is calculated by adding addresses in two registers and a displacement stored in the instruction. This addressing mode can be used to access an array of records.

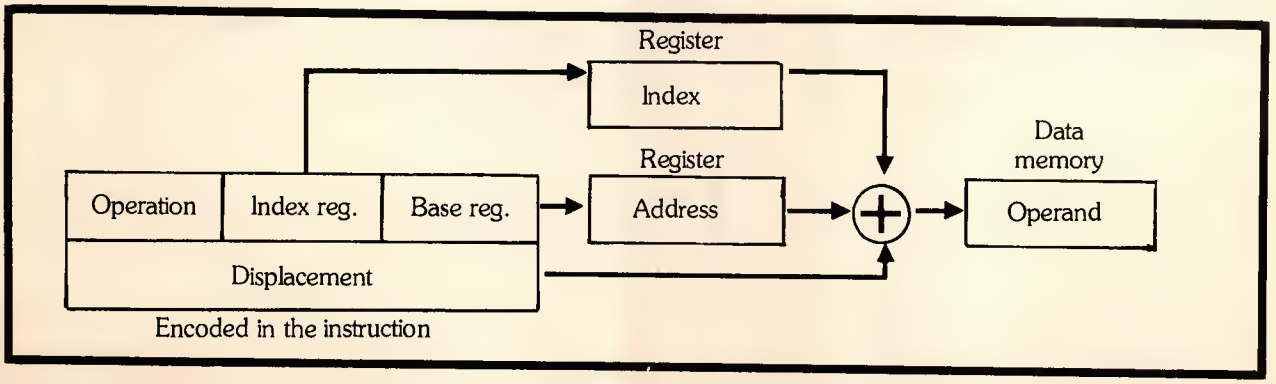


Figure 3. The base index with displacement addressing mode can be used to access subfields of elements in arrays of records. The base register holds the address of the first array element and the index register identifies the target element. The displacement points to the record field.

The base register will then contain the beginning address of the first array element. The index register will contain a scaled adder that, when combined with the base address, will give the starting address of the target record. The displacement value will then identify a field within the record.

If more complex addressing modes such as double indirection are required, they can be easily generated with the load address instruction. Using any of the standard addressing modes, this instruction calculates the effective address of the operand and loads it into a destination register, without affecting data.

The system interface

A number of architecturally advanced system bus interface control features are integrated into the Z80000. These features simplify hardware design and increase overall performance. The bus interface protocol of the Z80000 for memory and I/O operations is essentially identical to that of the Z8000 16-bit processor. Therefore Z8000 peripherals can interface directly to the Z80000, providing a ready-made set of support circuits. Since the Z80000 is architecturally compatible with the Z8000 at the application level, it comes with a ready-made library of support software. While applications in the 32-bit arena will be software-driven—with the majority of that software written in high-order languages and thus providing greater portability—this compatibility characteristic will provide valuable support for real-time applications.

As shown in the pin function diagram in Figure 4, the Z80000 utilizes a 32-bit multiplexed address/data bus with accompanying control and status lines for external memory and I/O accesses. The hardware interface control register is initialized to specify bus interface characteristics. The bus width may be designated as 16 or 32 bits. Bus clock speed and automatic wait-state generation are also specified. The clock input to the processor may range up to 25 MHz and is divided by two for clocking internal operations. The input clock is divided by either two or four for memory and I/O timing. This arrangement simplifies interfacing of the processor to slower memories and peripherals. For example, if the input clock is 24 MHz, memory and I/O operations are clocked at 6 MHz if the divide-by-four parameter is selected—this allows reasonably priced slower memories to be used in the design.

Automatic wait-state generation also simplifies hardware design. Up to seven automatic wait states

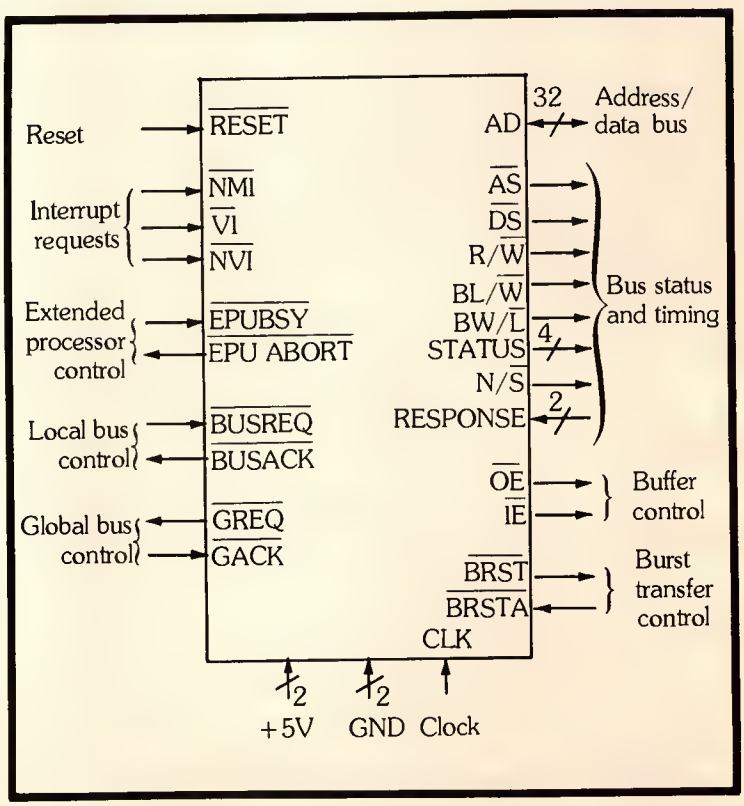


Figure 4. The 64 active pins of the Z80000 fall into nine functional groups supporting the external system interface. In addition to standard memory and I/O interfacing, multiprocessing, coprocessing, burst-mode transfers, buffer control, and DMA are directly supported.

can be generated in sections of either memory or I/O space. The output and input enable signals from the Z80000 can be used to enable address/data line buffers, further streamlining hardware design.

The Z80000 supports memory systems constructed with nibble-mode RAMs. If burst mode is enabled, then the processor issues a single address that may be followed by four consecutive data transfers. Without wait states, write operations require three clock cycles and read operations only two. If burst mode is in effect, these fast access times are further reduced, since generation of a single address is used for multiple transfers. The burst request and acknowledge lines are used with external hardware to define the areas of memory where burst mode is supported.

Multiprocessing is supported for local and global memory areas. The bus request and acknowledge signals support DMA sharing of the local bus, allowing interleaved access to memory and I/O space. The global request and acknowledge signals provide the protocol for sharing global memory areas with other general-purpose processors.

Coprocessing is supported by a set of special extended processing architecture, or EPA, instructions for data and command transfer. Coprocessors such as the Z8070 floating-point processor monitor the address/data bus, detecting the first word of an EPA instruction and participating in subsequent data transfers as required. A two-bit field in the instruction can identify up to four coprocessors concurrent in the system. Coprocessors can operate asynchronously from the main Z80000, performing assigned tasks while the Z80000 continues with other operations. The EPA busy and abort signals provide a synchronization capability. The abort signal can be used to terminate the current operation of the coprocessor. The busy signal is used by the coprocessor to halt the main processor if a command is issued before a previous command is completed.

Memory management

The major function of an operating system is to manage available resources so that system activities and application tasks can be made to accomplish their defined objectives in a timely and efficient manner. A major aspect of this task is memory management. The Z80000 incorporates an on-chip memory management unit (MMU) that supports a paged virtual memory addressing scheme with a page size of 1K = 1024 bytes.

There are two major concepts involved in memory management that lend themselves to direct hardware support. These are address representation and address translation. Address representation refers to the format used in doing address arithmetic to compute a logical effective address. Address translation refers to the action of converting this logical effective address into an address that can access a physical memory location in a particular hardware configuration. Thus the CPU's address representation and addressing modes determine the logical effective address, and address translation maps that address onto a physical memory location.

Two types of address representation, usually referred to as linear and segmented, are used on microprocessors. With linear addressing, addresses have a simple integer representation, and address arithmetic is based on the fundamental operation of addition modulo address size. With segmented addressing, computation of an effective logical address involves two separate values. One is referred to as the segment number and is unaffected by address arithmetic. Conceptually the segment number is a pointer identifying an object or block in memory. The other value is commonly referred to as the segment offset. Arithmetic is performed on this offset field only, and is based on the operation of addition modulo segment size. The segment number and offset are then combined to generate the logical effective address.

Zilog supports segmented addressing on Z8000 processors but offers both linear and segmented addressing as software-programmable options on the Z80000. This provides both compatibility with existing Z8000 application software and complete flexibility in generating new designs. Linear addressing can be selected in either full 32-bit or compact 16-bit form. The compact mode provides an efficient storage format that saves code space with addressing modes that require addresses to be stored in instructions. As shown in Figure 5, the two forms of segmented addressing provide segment sizes of either 64K bytes or 16 megabytes. The most significant bit of the address determines which address representation is in effect. If the most significant bit is one, then a seven-bit segment number is used. If it is zero, then a 15-bit segment number is employed.

The address translation mechanism of the Z80000 transforms the logical effective addresses produced by the address calculation logic of the CPU into physical addresses that actually reference elements in physical memory. As logical addresses are passed to the translation mechanism they are checked for proper access privileges and are translated if no violation occurs. If an infraction is

detected, the translation process is aborted without referencing memory, and the operating system is notified that corrective action is required.

The MMU of the Z80000 uses translation tables stored in general memory to define a correspondence between logical and physical addresses. This is the same technique as used by numerous mini/mainframe manufacturers, including IBM on the 370 series of computers.

As illustrated in Figure 6, the Z80000 supports up

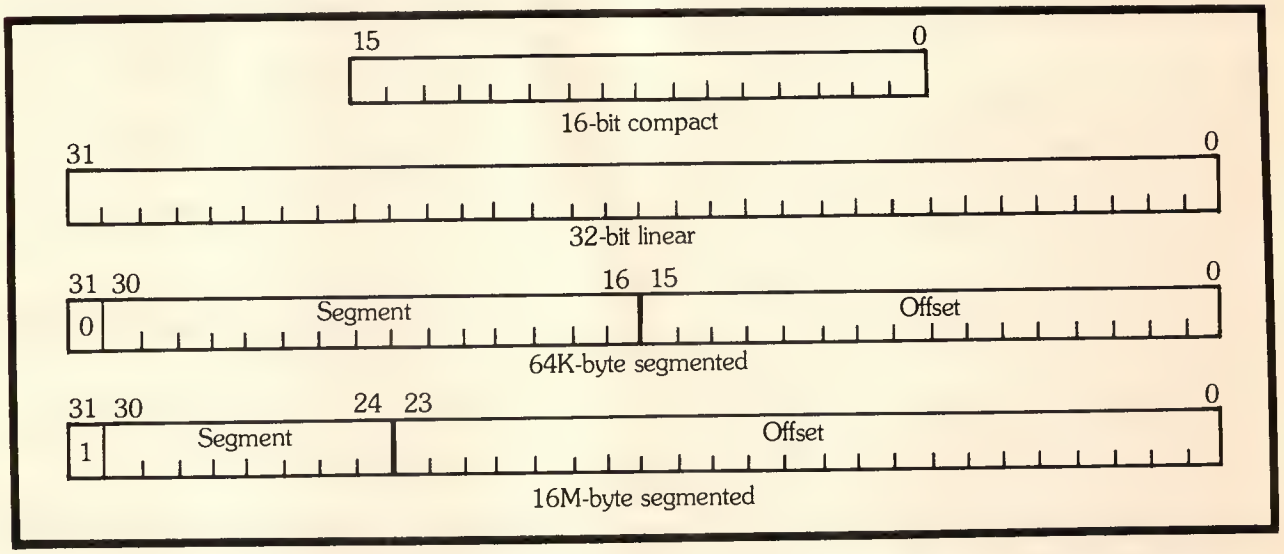


Figure 5. The Z80000 supports four address representations: 16-bit compact, full 32-bit linear, 64K-byte segmented, and 16M-byte segmented.

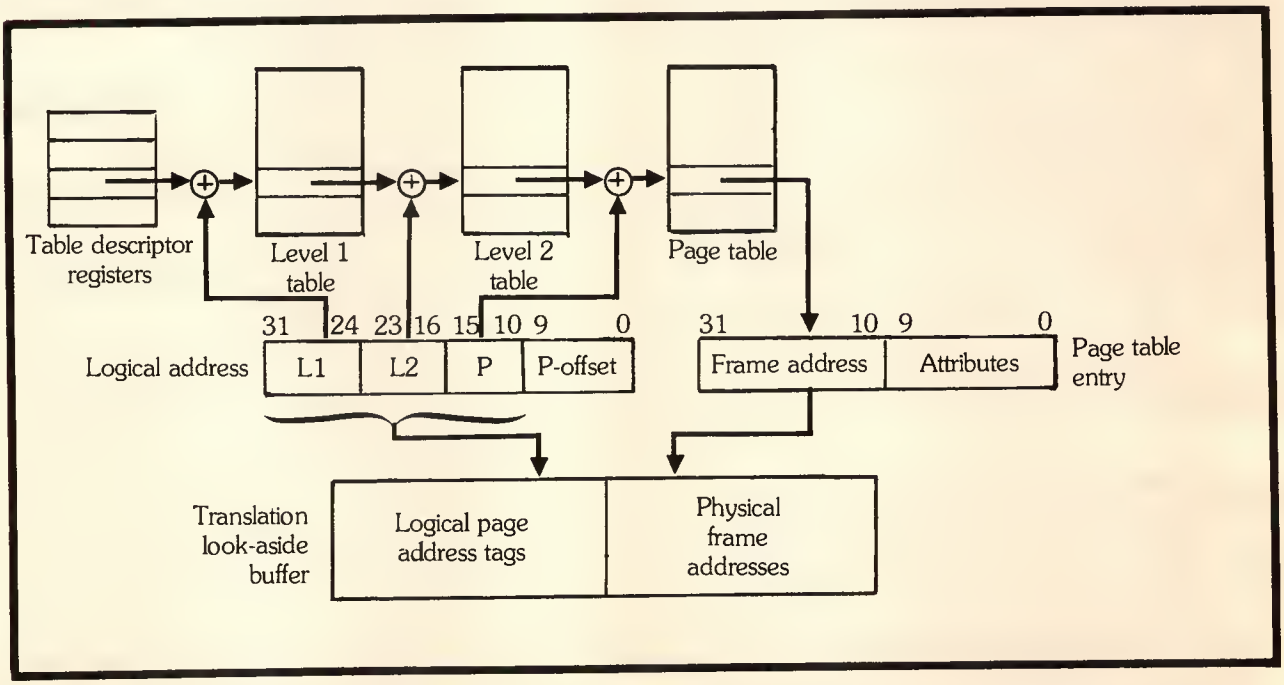


Figure 6. Address translation tables in memory define a mapping between logical and physical addresses.

to three levels of translation. A table entry at one level points to the beginning address of the next table. The translation hardware uses subfields within the logical address as pointers into these translation tables. Hardware pointer registers are initialized to point to the beginning tables, and calculation of a physical address involves a chaining from one table to the next. Since the Z80000 supplies external status information that allows physical memory to be divided into separate address spaces for system and normal modes and then allows each of these spaces to be subdivided again for code and data, separate tables and pointer registers are required for each of these four spaces that is supported.

The last tables used in the translation process are referred to as page tables. Page table entries contain the beginning address of the 1K pages in physical memory. In the translation process the most significant 22 bits of the logical address are used in the table-chaining process to identify a page table entry. The least significant ten bits of the physical address are combined with the page address from the page table entry to determine the physical address.

Translating a logical address into a physical address requires that table entries be read from memory. Reading table entries before generating the physical address imposes an unacceptable overhead, however. To overcome this problem, the Z80000 employs a small set of buffer registers implemented in a fast, fully associative cache memory. This set of buffer registers, referred to as the translation look-aside buffer, or TLB, holds the most recently used set of page table entries. It is checked before the memory tables are referenced. If the required translation value is in the TLB, translation time is drastically reduced. Since programs tend to operate locally for relatively long periods of time, most addresses can be translated from entries in the TLB.

The details of the translation process are determined by system configuration parameters dependent on the address representation and physical memory model currently in use. Three levels of translation are possible, and complete flexibility is provided. Translation can be disabled entirely, or selective levels can be skipped. For example, with Z8000 segmented addressing the segment number is right-justified in the first byte of the two-word address field. The second byte is ignored and the offset is in the following word. On the Z80000, the second level of translation uses the second byte of the 32-bit address as a displacement into the second-level tables. When the Z80000 runs Z8000 application software, the MMU is configured to skip the second level of translation.

Regardless of the translation configuration in effect, page table entries (Figure 7) have special significance for virtual memory support and system control. The valid bit indicates whether a page is resident in physical memory. The referenced and modified bits provide a history of access information that can be used by the operating system to implement a paging

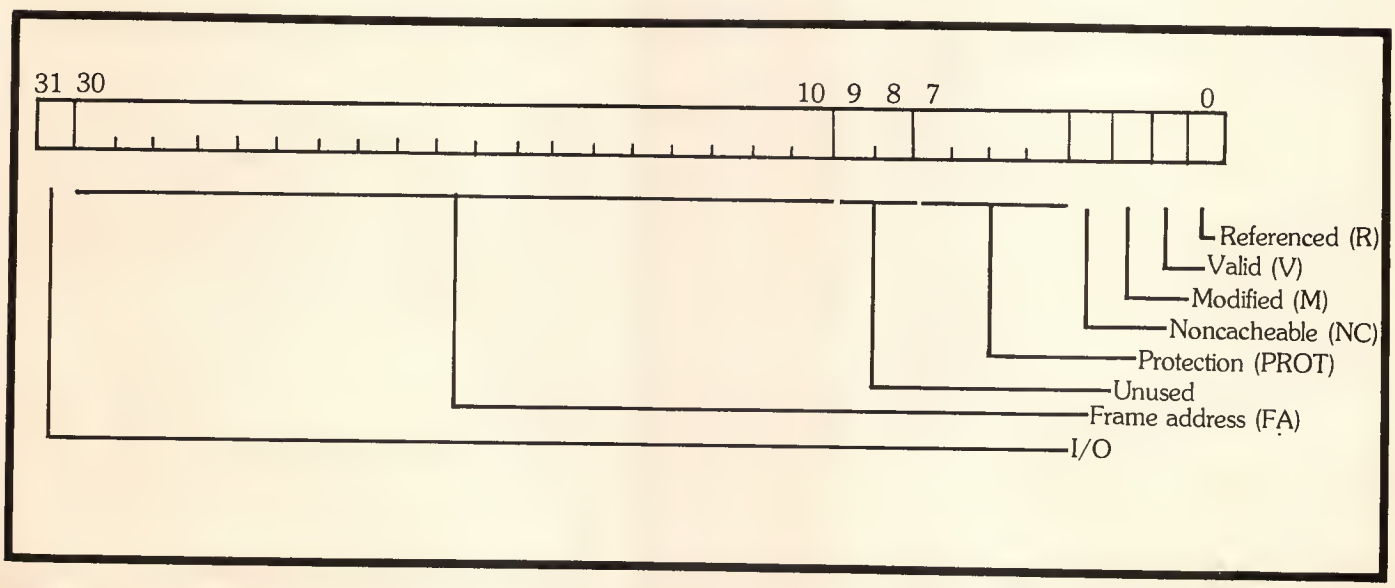


Figure 7. Page table entries contain referenced, valid, and modified bits for virtual memory support. Other fields indicate whether page elements can be stored in cache and the protection characteristics of the page. The I/O bit determines whether the page is in memory or I/O space. The frame address is the beginning address of the page.

algorithm for virtual memory support. On the first access to a valid page table entry, the referenced bit is set. On the first write to any element in a valid page, the modified bit in the page table entry is set.

The cacheable bit can be used in multiprocessing systems to specify shared memory areas as noncacheable. The I/O bit allows a page to be placed in memory-mapped I/O space. If the I/O bit is set, then a read or write operation generates the bus-control signals of an explicit I/O instruction.

When a logical address is presented to the memory translation unit, an associative look-up operation compares the bits not in the page offset field to entries in the TLB. If no match is found, then the table-chaining process is performed to do the translation, and the resulting page table entry is automatically stored in one of the TLB registers. Choosing which TLB register value to replace is critical to system performance.

The register replacement scheme used on the Z80000 is based on the least recently used, or LRU, algorithm. With this algorithm, the registers in the TLB are ordered as in a stack according to access history, with the least recently used register on top. When a TLB register is required for a new entry, it is that register that is used.

The 16 page-table entries of the TLB represent 16K logical addresses that can be translated directly from the TLB. Since programs tend to operate locally over time, most logical addresses will be translated from the TLB. A TLB miss imposes an overhead of up to three table accesses and causes a page table entry to be loaded into the TLB. While the TLB hit ratio is dependent on the hardware configuration of the system, the number of translation levels in effect, and the software being run, simulations have shown that a TLB hit ratio of 98 percent is common.

The Z80000's MMU simplifies a number of operating system functions. In addition to providing instruction abort and restart for virtual memory support, its multiple table structure provides a simple means to define valid address ranges. Because the Z80000 has a four-gigabyte addressing space, it is important to provide some means to prevent access to addresses that are not implemented in physical memory. With the Z80000's MMU this can be accomplished by in-

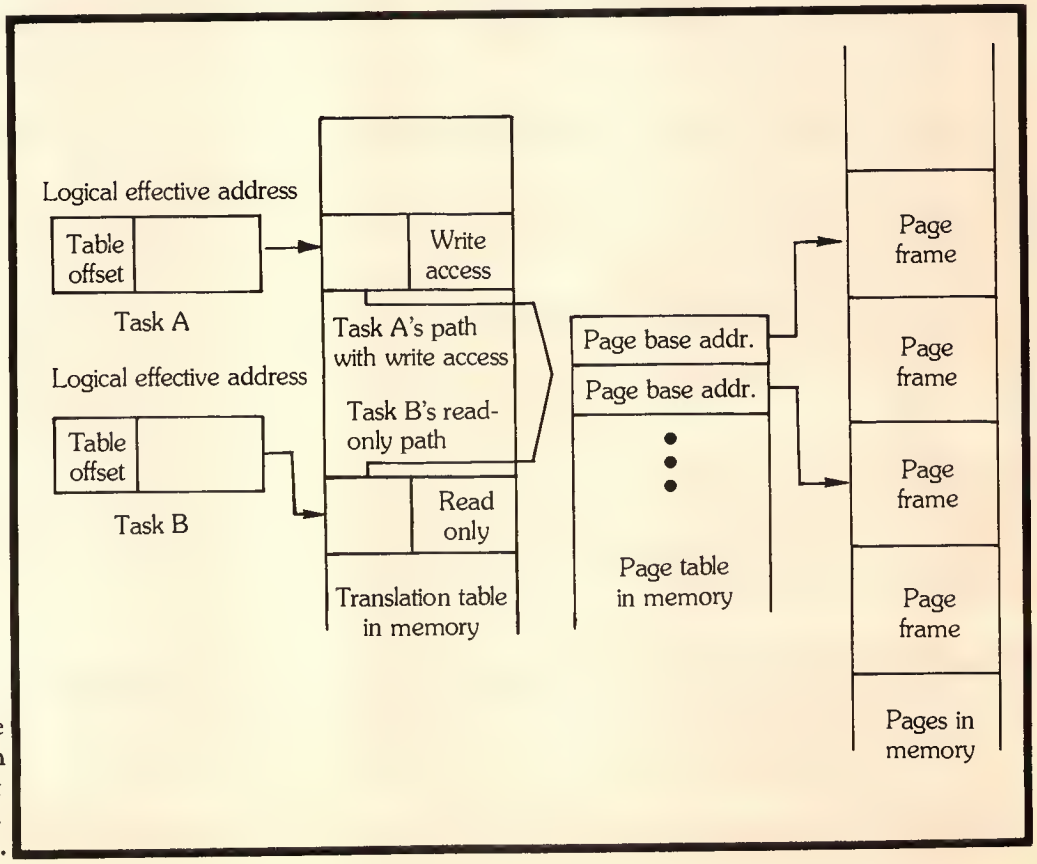


Figure 8. Separate translation paths can provide different access privileges to two tasks.

validating a table entry at any of the three levels. Access to an invalid entry causes a system trap.

The MMU provides a second type of protection for valid table entries. At any translation level, read, write, and execute privileges can be specified for both system and normal mode accesses in a number of combinations.

Another operating system function simplified by the MMU is the context switch. If a switch from one task to another requires a new logical-to-physical address mapping, the MMU can accomplish it simply by purging the TLB and reinitializing the appropriate table pointer registers to point at a new set of translation tables that have been previously initialized.

In addition to these explicit block-level protection features, address translation in and of itself provides the operating system with an important implicit protection mechanism. The operating system can absolutely control the physical memory that a task can access by controlling the setup of the translation tables and pointer registers before the task is activated.

A final form of protection that is provided by multiple translation levels involves access privileges to data by tasks having separate translation paths. Once protection is specified in a table entry at any translation level, all pages with paths through that table entry inherit the specified protection. This protection structure allows separate table entries to be mapped onto the same set of pages, but with different access protections. As shown in Figure 8, this technique can be used to provide separate protection paths to a database for different tasks.

Loosely and tightly coupled multiprocessor systems

For the designer seeking maximum system performance, multiprocessing support is an absolute necessity. For example, multiprocessing support in the form of DMA operations has become the standard technique for handling data transfers from mass storage to memory.

In general a multiprocessing system is one in which more than one processing unit is available to execute programs or perform system tasks. The goal of such configurations is to increase total system performance and/or reliability. The key to the successful application of these configurations is interprocessor communication and resource allocation.

Systems containing multiple general-purpose processors can be classified in terms of their memory configurations as either systems that contain shared memory or systems that do not. Systems with shared memory are said to be tightly coupled, whereas those without shared memory are said to be loosely coupled. In loosely coupled systems, interprocessor communication is accomplished via an I/O structure.

The performance requirements of many applications for the Z80000 will require multiprocessing in a tightly coupled configuration. The Z80000 provides direct hardware support for this requirement by providing external signals that allow memory to be subdivided into local and global areas. For tightly coupled systems, it furnishes two buses to each processor. A local bus provides access to the local memory area that is unique to the processor, while a global (system) bus provides access to the global memory resource. Arbitration logic comes into play only for global accesses, allowing multiple processors to operate concurrently from their local memory. In global/local configurations, the global bus will be a timeshared common bus.

The hardware interface configuration register allows the system programmer to define certain address ranges as "global." When an address in the global range is accessed by an instruction, the global request (GREQ) and global acknowledge (GACK) lines on the processor are used to implement a global memory protocol as follows: The processor outputs valid address and status information and then drives the GREQ line active (Figure 9). When an active GACK signal is returned, the processor proceeds with a normal memory transaction. The global memory arbitration logic, which is external to the processor,

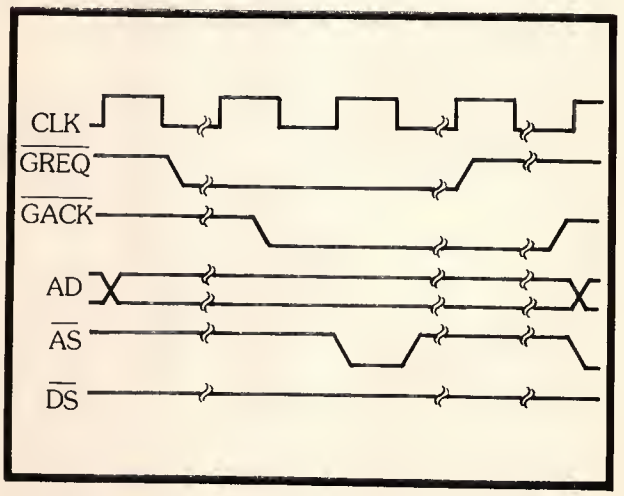


Figure 9. In systems with global memory, global request and acknowledge lines are utilized by external arbitration logic to control access to the global memory areas.

utilizes the GREQ and GACK lines of the processors in the system to regulate access to the shared memory.

Global memory is regulated by two fields contained in the hardware interface configuration register. The global enable (GE) bit enables/disables global memory transactions, while the four-bit local address (LAD) field indicates the valid range of local addresses. When a memory or I/O access is initiated and GE = 1, bits in the LAD field are compared with bits 26 through 29 of the physical address. If they are equal, then a local transaction is performed. Otherwise, a global transaction is generated (Figure 10). This arrangement allows one of 16 memory areas to be designated as a processor's local memory, with the rest of memory representing either global memory or local memory for other processors.

A classic difficulty often referred to as the "cache coherence problem" arises in multiprocessor systems that employ cache memory. If one processor reads data from shared memory and stores that data in its cache buffer, then subsequent reads may come from that buffer rather than global memory. If another processor has written that data in the meantime, then incorrect values may be processed. Since most systems of any size employ multiprocessing in some form (such as DMA), some means must be provided to deal with the cache coherence problem.

The Z80000 provides tools sufficient to manage the cache coherence problem without drastically affecting system throughput. The memory management unit allows individual pages to be specified as noncacheable. Since memory management stands between the execution unit and cache, references to noncacheable pages are directed to physical memory, bypassing cache. Global areas of memory that are not write-protected should be located in noncacheable pages. Read-only global data areas and strictly private local data areas (no DMA access permitted) may be located in cacheable pages. If DMA operations occur in local memory, a page may be designated as noncacheable during periods when DMA may occur and as cacheable otherwise. However, if a processor opens a page for DMA access, it must flush the cache buffer to ensure that later references will be current.

The Z80000 provides a set of special interlocked instructions that can be used for interprocessor communication. These instructions generate atomic read/modify/write operations that can be used as synchronization primitives for semaphores. When one of these instructions executes, the bus-request input signal is not recognized until the instruction completes, thereby locking out interleaved local memory accesses. In addition, a unique external status is provided to indicate that an interlocked instruction is being executed. If a semaphore resides in global mem-

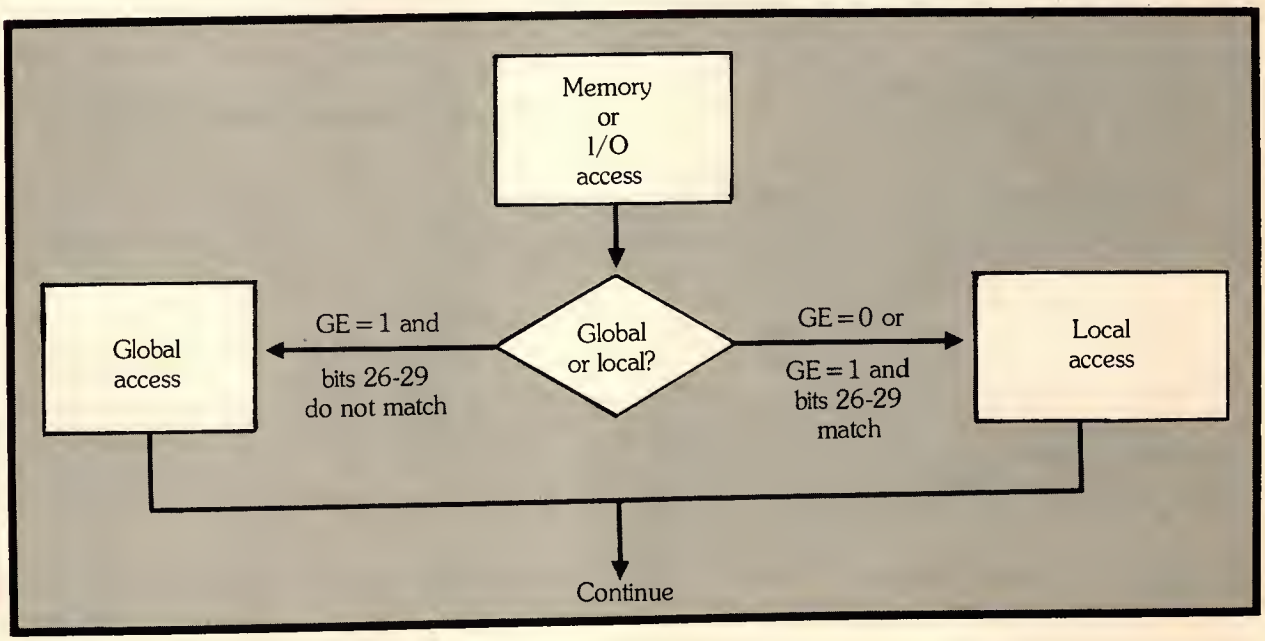


Figure 10. Fields in the hardware interface configuration register define the address ranges that generate the global memory access protocol.

ory in a tightly coupled system, the arbitration logic controlling shared memory can utilize this status information to disable interleaved accesses to global memory from other processors.

Test and set, increment interlocked, and decrement interlocked are the three interlocked instructions. The test and set instruction copies the most significant bit of the destination operand into the sign bit of the flag and control register and then sets all bits in the destination to logic one. With the other two instructions, the addressed memory location is either incremented or decremented by a specified value from one to sixteen. With the test and set instruction, the read operation is performed directly from memory, always bypassing cache. This feature prevents the obvious problem that could arise if a semaphore variable were read from this on-chip buffer.

Cache memory

A cache memory is a rapid-access local storage buffer that stands between the processor and main memory. Some memory accesses cause values to be stored in cache so that subsequent read operations to those same locations will be able to access cache only and will not require an external bus cycle. When the Z80000 fetches from cache, only one system clock cycle is required; when it fetches from off-chip memory, two or more cycles are required. Thus cache can decrease the time required to execute memory cycles, and it can free the bus for DMA or multiprocessing transfers, increasing the effective bus bandwidth of the system.

Cache memories are organized in multibyte blocks (called tag lines) with associated tag fields. The bus structure and internal organization of the processor determine the best block size. When cache is accessed, a comparison is made between valid tag fields and a subset of the address bits. If a match is found, additional bits in the address identify an individual entry (byte, word, or long word) in the tag line.

As shown in Figure 11, the Z80000 has a 256-byte, fully associative cache organized as 16 tag lines, each containing 16 bytes managed as eight words. "Fully associative" means that any tag line can be associated with any address block. Since the Z80000 supports burst-mode transfers, an entire tag line of 16 bytes can be loaded by the four consecutive 32-bit transfers of a burst read, making the 16-byte block size a logical choice. The cache can be configured to hold instructions only, data only, or both instructions and data. The cache can also be frozen to hold values from a dedicated memory range, to provide very fast access to either a critical subroutine or important data.

The addresses represented by the tag line are determined by the value stored in the tag field. When a cache access is attempted, the 28 most significant bits of the physical address are simultaneously compared with all valid tag fields. If a match is found, then the least significant bits point to the target word in the cache line.

In addition to the 28-bit tag field, each tag line contains a validity bit indicating whether the address tag field is valid. It also contains an eight-bit field that indicates whether the individual words of the cache line are valid. The 16 bytes of a tag line are aligned on a 16-byte boundary and addressed sequentially. Thus a cache line consists of a 28-bit tag field and a validity bit, an 8-bit validity field, and a 16-byte (8-word) storage block. When a memory reference is made, the 28 most significant bits of the 32-bit physical memory address are simultaneously compared with all cache line tags. If a match is found and the valid bit is set, then bits one to three of the address identify one of the eight words in the block. The validity bit for that word is then checked. If it is set and the operation is a read, then the access is from cache. Otherwise memory is accessed and the

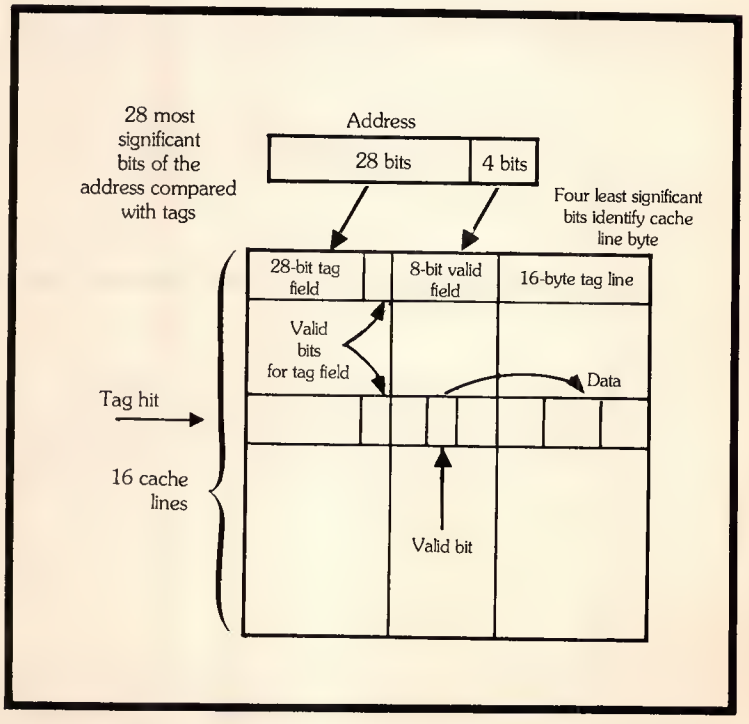


Figure 11. The 256-byte cache is organized as 16 tag lines of 16 bytes each.

cache line can be simultaneously and automatically updated.

It should be noted that a cache miss may be either a tag miss or a tag hit followed by a word miss. If a tag miss occurs and the following memory access requires that the cache be updated, then the fully associative nature of the cache requires that one of the cache lines be reallocated. The LRU algorithm is used to choose the line to be replaced.

An important consideration that affects the performance of the cache has to do with the way write operations are handled in cases in which the cache holds data that may be modified. Read operations are straightforward—a value is retrieved from the cache if it is stored there, or it is read from memory and the cache is updated if the address is cacheable. Write operations, however, present a number of possibilities for updating the cache. With write operations on the Z80000, both the cache and memory are updated in the case of a tag hit. Called the "write through technique," this scheme ensures that memory and the corresponding cache data entries always agree. If a tag miss occurs on a write operation, only memory is updated.

The performance increase that is realized from cache operation is dependent on the cache hit ratio, defined as the probability that a given memory reference can be found in the cache. The cache hit ratio is a complex function of several system hardware and software variables and is typically estimated statistically by running software application samples. The system performance increase realized from cache operation can be expressed as a function of the cache hit ratio. Simulations using the instruction mix generated by a C compilation of the UNIX Visual Editor indicate that a cache hit ratio in the 60 to 70-percent range may be expected. Such a cache hit ratio should produce an overall performance increase in the 20- to 30-percent range.

Pipelining

Pipelining increases performance by allowing a processor to execute multiple instructions in parallel. This feat is accomplished by dividing each instruction into basic operations and dedicating individual execution units to each one. As shown in Figure 12, the Z80000 divides each instruction into six stages of execution: instruction fetch, instruction decode, operand address calculation, operand fetch, execution, and operand store. No instruction can advance to a stage in the pipe until its immediate predecessor has completed that stage and released the execution unit. When an instruction completes, the instruction just succeeding it may also be partially completed. Under optimal conditions, an instruction will complete as a concurrent set of pipeline stages completes, so that one instruction will enter the pipe as another exits. Under these conditions, processor performance may increase by a factor of six. Unfortunately, the following factors work against this performance increase:

- Events such as jump or call instructions, interrupts, and traps that cause a new program counter value to be loaded may cause the pipe to be partially or fully flushed.
- Some instructions do not use all stages of the pipeline; hence, execution units will occasionally be idle.

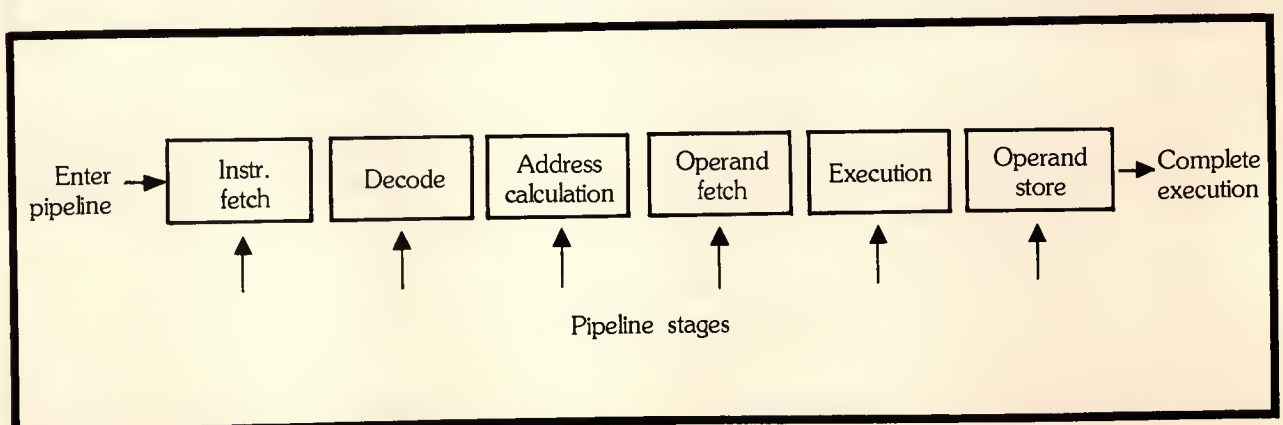


Figure 12. The Z80000 uses a six-stage pipeline for instruction execution. Separate ALUs are provided for address and operand calculation. Under optimal conditions, an instruction will complete as a set of concurrent pipeline stages completes.

• Some stages of the pipe require more time than others, and the stage that requires the longest time to complete determines the rate at which instructions move through the pipe.

These factors make predicting the actual performance increase that will be realized quite difficult. However, nonsequential events are typically infrequent enough that the pipelining scheme of the Z80000 should produce a significant performance increase.

Architectural relationships among caching, pipelining, and memory management

The Z80000 should have remarkable performance capabilities. Simulations based on the instruction mix produced from a C compilation of the UNIX Visual Editor indicate that raw performance levels of up to 3.5 million instructions per second may be realized. Even assuming worst-case parameters, we can expect performance levels exceeding one MIPS. The Z80000's high levels of integration and performance require that the designer adjust his perception of the microprocessor-based system.

With 8- and 16-bit microprocessor designs, the pacing consideration often reduces to the question, "Can the job be done given these constraints?" Constraints may involve execution speed, board space, power consumption, or production cost. Extending performance beyond a certain required minimum is a luxury not usually afforded by processor power. The designer has a good idea of the performance limitations of his processor, and many applications require that he design to those limitations.

Given the capabilities of the Z80000, things will surely change. When optimized to target applications, Z80000-based systems will closely approximate the capabilities of mainframe computers, requiring a design model that transcends the simpler hardware and performance considerations of the past. While hardware will represent a relatively known quantity, software and its subtle interaction with the integrated units of the processor and the rest of the system will be less known and yet will be the determining factor in a system's performance and in its total effectiveness over the long run. Therefore, the designer should know the architecture of the processor he has chosen and how its internal structure will affect performance for his target applications.

We have surveyed some of the architectural features of the Z80000 that will contribute to its high performance. In addition to having a functional understanding of these features, the designer should have some regard for their interaction. The compact, functional integration of cache and memory management onto the microprocessor not only provides increased performance but allows these functions to be set up so that they have the most advantageous relationship to one another. Figure 13 shows a functional block diagram of the execution unit, memory management unit, and execution cache of the Z80000. An important aspect of this configuration has to do with the relationship between the MMU and the cache. Since the MMU stands between the CPU and the cache, physical addresses are used to determine cache hits. When the processor issues a logical address in order to access memory, that address is checked against entries in the TLB. If a match is found, the resulting physical address is compared against entries in the execution cache. If a TLB miss occurs, the MMU determines the physical address from translation tables in memory and then compares it against entries in the cache buffer.

While the operation of the pipeline, the cache, and the MMU should be relatively automatic once the operating system configures the processor state and the translation tables, some care must be taken in the setup process. Consider, for example, the relationship between the pipeline and the MMU. During the process of either enabling or disabling the MMU by loading the configuration register, invalid instructions may be prefetched into the pipe. It is the responsibility of the operating system to ensure that the results of such instructions are benign.

The Z80000 will achieve performance levels previously reserved for mainframe computers. Its integration of a cache, a multistage pipeline, and a memory management unit will result in high effective bus bandwidths by significantly reducing the time used in performing memory transactions. By integrating into a chip the major parts of a large system and by providing performance levels of from one to five MIPS, the Z80000 will open a new world of applications to the microprocessor system designer. At the same time, it will encourage the designer of mainframe computers to take a new look at microprocessors as a possible design path for his future products.

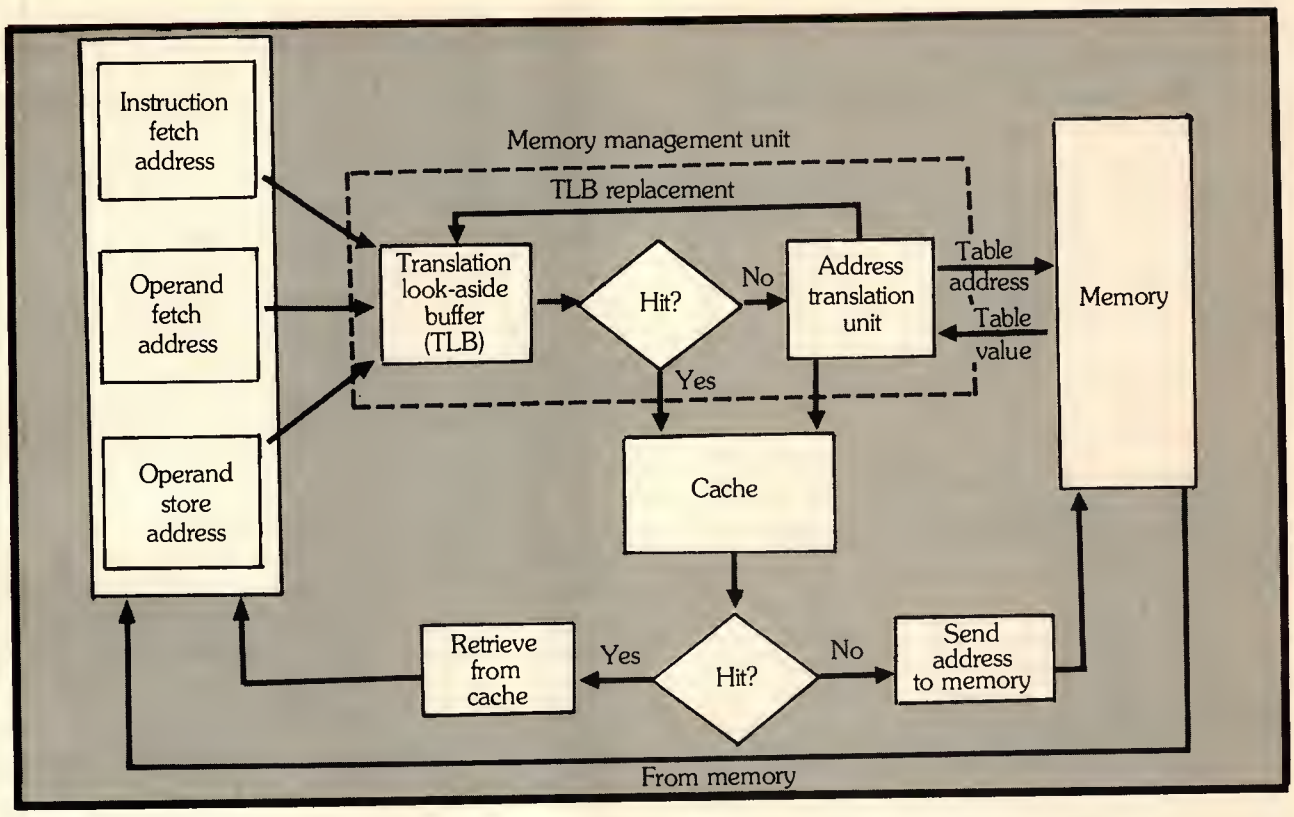


Figure 13. In the translation configuration of the Z80000, the MMU stands between the CPU and the cache. Because of this arrangement, physical addresses are used to determine cache hits.

Bibliography

Alpert, D., "Powerful 32-bit Micro Includes Memory Management," *Computer Design*, Oct. 1983.

Denning, P., "Working Sets Past and Present," *IEEE Trans. Software Engineering*, Vol. SE-6, No. 1, Jan. 1980.

Eibner, J. A., "Simple Cache Steps Up Performance of 16-bit Systems," *Electronic Design*, Dec. 22, 1983.

Johnson, R., "Microsystems Exploit Mainframe Methods," *Electronics*, Aug. 11, 1981.

Mudge, J. C., "Design Decisions Achieve Price/Performance Balance in Mid-range Minicomputers," *Computer Design*, Aug. 1977.

Phillips, D., "Microprocessor Manages Virtual Memory for Large Programs," *Electronic Design*, Oct. 27, 1983.

Phillips, D. F., "Memory-Management Varieties Suit Different Application Areas," *EDN*, Sept. 6, 1984.

Phillips, D. F., "Mainframe Tricks Raise Performance of 32-bit Micros," *Computer Design*, July 1, 1985.

Ramamoorthy, C. V., and H. F. Li, "Pipeline Architecture," *Computing Surveys*, Vol. 9, Mar. 1977, pp. 61-102.

Schmitt, S., "Virtual Memory for Microcomputers," *Byte*, Apr. 1983.

Z80000 Preliminary Technical Manual, Zilog, Inc., Campbell, CA.

David Phillips is manager of field applications engineering at Zilog, Inc. A graduate of the University of Texas, Austin, he obtained his MA in mathematics and philosophy and his PhD in topology, both from the University of Texas. He taught mathematics at the University of Georgia and at the University of New Mexico prior to joining the semiconductor industry.

Questions about this article can be directed to Phillips at Zilog, Inc., Mail Stop C2-5, 1315 Dell Avenue, Campbell, CA 95008.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 159 Medium 160 Low 161

ITT CAP—Toward a Personal Supercomputer

Steven G. Morton, Enrique Abreu, and Fred Tse
ITT Advanced Technology Center

As microprocessor technology has advanced, greater and greater computational power has migrated down to the level of the personal computer. Today we can speak of the "personal minicomputer," and many even speak of the "personal mainframe." But is a "personal supercomputer" within reach anytime soon? ITT thinks it is, and tc that end the company initiated the Cellular Array Processor Project at its Advanced Technology Center in January 1982. The objective of the CAP Project is to design a processor providing performance far beyond what is economically feasible with processors using commercially available parts. The CAP will be applied particularly to digital signal processing and image processing as well as to engineering analysis and database manipulation. Its performance will be able to be scaled to an application, and it will be able to be used as an embedded or attached processor.

The CAP employs a highly parallel, highly regular design based on a single-instruction, multiple-data stream, or SIMD, architecture. It is being built from large, custom CMOS chips that are economically feasible due to their fault-tolerant design. The principal chip in the CAP—the array chip—is a generic "vector" bit-slice component that should also find use in many dedicated applications. The array chip has a simple, regular design that allows all elements, including a few spares, to be controlled by software. In this way, words of varying sizes can be formed, manufacturing defects can be overcome, and lifetime failures can be compensated for. A planned controller chip will drive an array of array chips and provide a reduced instruction set for parallel processing.

We should note that there are two versions of the array chip—Array Chip I and Array Chip II-M. Array Chip I is our engineering prototype and contains 20 one-bit-wide bit slices. Array Chip II-M, which we are still developing, will be the basis for the personal supercomputer described here. Array Chip II-M will contain twenty 16-bit-wide bit-slice processors and a substantial amount of dynamic RAM, all in one chip.

The personal supercomputer is actually a coprocessor for an IBM PC-AT or compatible. Since it is based on Array Chip II-M, it is still a paper machine. Hence, all performance figures we cite are *targets*, not measurements. We are planning product introduction for 1987.

In the following sections, we present our rationale in designing our systems the way we did. First, we explain why we chose the SIMD class of architecture and how our design compares to other SIMD machines. We discuss the importance of fault tolerance in implementing our VLSI chips. We then describe our architecture, in terms of both hardware and the programming model. Finally, we review the characteristics of our first-generation prototype, which serves as our feasibility model.

Why SIMD?

Why did we choose a SIMD architecture? Simply because the performance of an SIMD machine for a given cost is superior to that of any other architecture for the applications we are trying to serve. The highly regular nature of our problems fits the highly regular design of our machine. And that highly regular design—based on central control, regular processing

0272-1732/85/1200-0037$01.00 © 1985 IEEE

elements, and regular communication paths between them—is the key to the SIMD machine's very low cost/performance ratio.

Many excellent articles on the various types of parallel processors have appeared recently. The June 1985 issue of *Computer*[1] was an excellent survey of the field, and the July 1985 issue of *High Technology*[2] examined parallel processors from a business point of view. We will therefore be brief in our overview.

The target applications of the ITT CAP—digital signal processing, scientific and engineering processing, and database manipulation—exhibit certain common characteristics. In these applications,

- data structures are highly regular,
- data elements are processed in large blocks,
- the volume of input data is very large,
- the desired response time is often very short and critical, and
- the computation requirements per datum are relatively uniform.

The SIMD architecture is well suited to such regular applications. It is inherently highly structured and can be configured into different sizes without much overhead.

Multiple-instruction, multiple-data stream, or MIMD, architecture is well suited to tasks characterized by randomness. Irregular data structures or irregular computational tasks provide such randomness. While MIMD machines could handle our regular applications, the cost/performance they would provide would be orders of magnitude worse than that for SIMD machines.

The single-instruction, single-data stream, or SISD, architecture—the architecture of microprocessors—was not our choice because our performance target would have required very high clock rates that could not have been achieved even with expensive, power-hungry, low-density bipolar circuits. Perhaps gallium arsenide or another exotic technology will eventually solve some of the problems we are interested in, but they cannot help us now.

The multiple-instruction, single-data stream, or MISD, architecture is exemplified by systolic arrays, which are so far largely special-purpose in nature. However, the architecture we describe here—for certain classes of algorithms and vector inner loops—may be considered a programmable systolic array.

The SIMD class includes both array processors and cellular array processors. An array processor generally has a high-performance pipeline of arithmetic elements, exhibits little parallelism, and operates on an array of data. A cellular array processor—which we are building—is highly parallel: it has an array of identical processors, with each processor in the array operating on an array of data. The multiplicity of processors lends itself well to highly structured VLSI design techniques, especially when those techniques are extended by procedures for providing fault tolerance.

Other SIMD machines and the CAP. There are numerous examples of array processors, but only a few cellular array processors. The Goodyear MPP—Massively Parallel Processor[3]—and the ICL DAP—Distributed Array Processor—are the most often mentioned machines in this class, and the recently developed NCR GAPP—Geometric Arithmetic Parallel Processor—chip has begun to receive attention. Although the ITT CAP is similar to these machines in many ways, it also differs from them in several important respects:

- The CAP stores and processes the data stream differently. The MPP, DAP, and GAPP are designed to operate in a bit-serial, word-parallel fashion. They store each word bit by bit through a succession of memory locations. The ITT CAP, like the classic Illiac IV, operates in a bit-parallel, word-parallel manner. It provides a more flexible method of memory addressing and a simpler programming model than the other machines.
- The CAP has an instruction set that directly supports high-level-language constructs such as if...then...else. The instruction set is simple and uniform. The CAP is inherently a parallel reduced instruction set computer.
- Due to both improved technology and fault-tolerant design, the CAP has many 16-bit processors and large amounts of memory on a single chip. The other machines integrate fewer processor bits with much smaller amounts of memory on a chip.
- For a given number of total processor bits, the CAP is better suited to engineering applications than the DAP or MPP, since the number of processors it provides is a better match to the dimensionality of the problem. Where the DAP handles 2^{12} variables bit-serially, requiring an enormous matrix for efficient use of the machine, the CAP handles 128 32-bit variables in word-parallel fashion, needing only a much more normal-sized matrix.

The ITT CAP, because of its later introduction and more advanced technology, should provide more than an order-of-magnitude better cost/performance than any other machine, and should be more reliable.

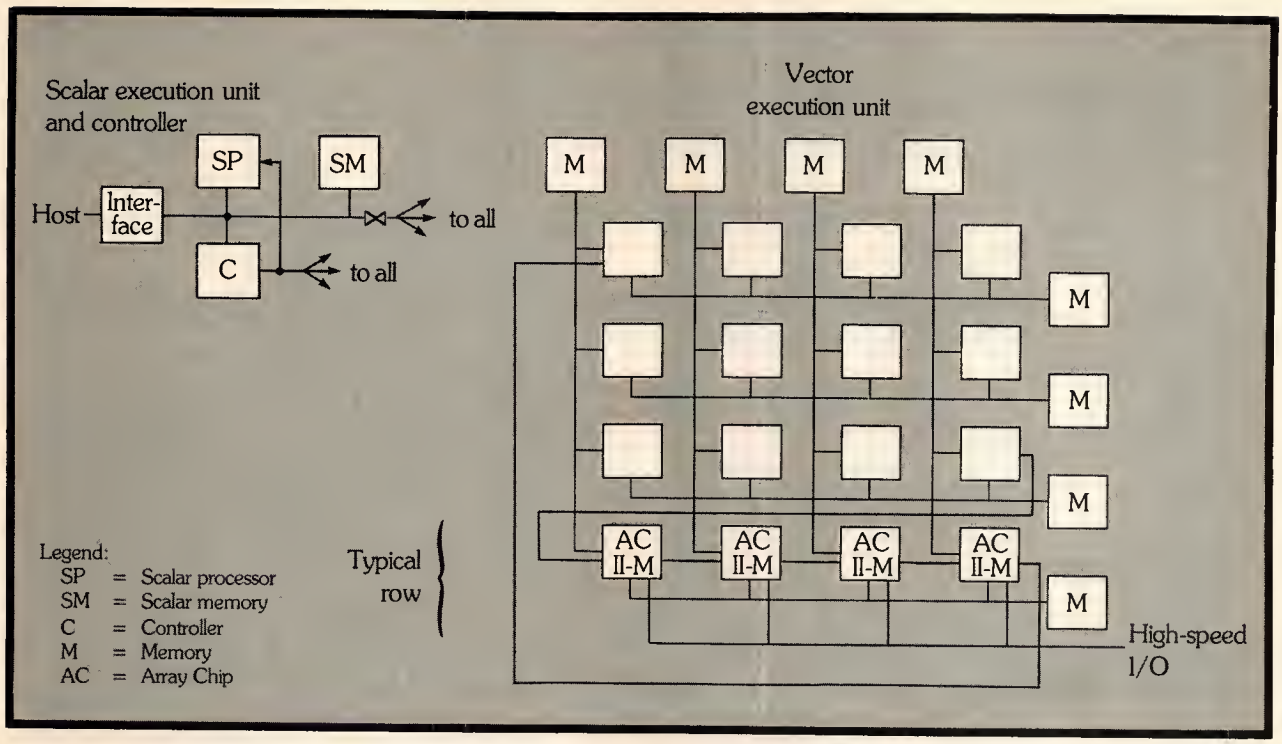


Figure 1. Block diagram of the ITT CAP personal supercomputer coprocessor.

Fault tolerance in VLSI design

There are many advantages to putting as many elements as possible on a single VLSI chip:

- It reduces system size and cost (the system requires fewer chips).
- It improves system performance (more elements on fewer chips results in shorter, faster communication paths).
- It increases system reliability (fewer chips means fewer mechanical connections).

However, there are also many problems associated with such practice.

By far the biggest obstacle to the production of very large chips is low yield—as die size increases, yield rapidly decreases. We decided to tackle this problem by following the strategy employed in the design of large memory chips—dividing the system into small blocks so that each block has sufficiently high yield, and including spare parts. In this way, faults in individual blocks can be tolerated.

Specifically, the strategy requires

- regular, cellular, structured processing elements,
- simple, small processing elements with high yields,
- minimal interconnections between processing elements,
- a small amount of critical logic in each processing element,
- spare processing elements,
- a bidirectional bypass mechanism,
- word configuration and processing element activity controllable by software, and
- a small amount of common logic on each chip.

These requirements are compatible with the SIMD architecture of the CAP. By meeting all of them—i.e., by designing for fault tolerance—we were able to build chips substantially larger and more powerful than those we would have been able to build without doing so.

Hardware architecture

The ITT CAP personal supercomputer coprocessor will contain a controller, a scalar execution unit, and a parallel, or vector, execution unit (Figure 1). Array Chip II-M's will be arranged in a 4×4 matrix, providing the user with 256 16-bit dynamically reconfigurable processors. The target specification for this CAP configuration is shown in Table 1.

Table 1.
Target specification for the ITT CAP supercomputer coprocessor.

Performance (peak)
- ⅔ that of a Cray-1
  - 120+ MFLOPS at 32 bits
- ⅓ that of the Massively Parallel Processor (MPP)
  - 1+ BIPS at 32 bits
  - 2+ BIPS at 16 bits
- 100-megabyte-per-second I/O bandwidth to external devices
- 12.5-MHz base clock rate

Architecture
- Highly parallel
- SIMD parallel RISC
- 256×16-bit/128×32-bit to 16×256-bit vector processors
- 32-bit scalar processor

Memory
- 256 kilobytes scalar memory
- 512 kilobytes total local vector processor memory
- 2 or 8 megabytes total external vector processor memory

Physical construction
- Single-board plug-in to IBM PC-AT or compatible
- Surface-mount technology with components on both sides of board

Cost/performance
- $5/MIPS at 16 bits
- $10/MIPS at 32 bits
- $80/MFLOPS at 32 bits

The controller provides instruction fetch and decode logic and a program counter. It controls the scalar execution unit and the vector execution unit.

The scalar execution unit contains the scalar address processor, the scalar data processor, and the scalar memory. The scalar memory contains both the scalar data and the instructions. All paths in this unit are 32 bits wide.

The vector execution unit contains multiple vector address processors, multiple vector data processors, and a hierarchy of vector memories. Most of the buses in this unit are 32 bits wide.

The address and data processors in an execution unit operate concurrently to improve performance. Each of these processors contains 16 registers and a 16-function ALU, and each of the vector data processors also contains a locally addressed DRAM and conditional execution (vector if...else) logic.

The scalar address processor handles the storage and manipulation of scalar address pointers. It can access the scalar memory and the horizontal and vertical vector memories. The scalar data processor handles the storage and computation of scalar data.

Likewise, the vector address processors handle the storage and manipulation of vector address pointers referencing the local, horizontal, and vertical vector memories. The vector data processors handle the storage and computation of vector data; they can access the external vector memories and their own local memories.

The Array Chip II-M's are connected in rows to horizontal memories and in columns to vertical memories. These memories are accessed in a time-division-multiplexed fashion by the vector processors. Row-to-row and column-to-column data transfers are also done over these connections. This architecture is particularly well suited to matrix manipulation, since for matrix multiplication a row element can be broadcast to all columns, or a column element to all rows.

In addition, vector processors can communicate with their left and right neighbors. The communication is concurrent for all vector processors and winds its way through the entire array.

Each row has a high-speed I/O bus that operates concurrently with processing and memory referencing. Its operation is analogous to that of a video DRAM.

Array Chip II-M. Figure 2 is a block diagram for the Array Chip II-M, here with 256K usable bits of on-chip dynamic RAM. The chip's key elements are the twenty 16-bit processors, or cells, each of which is coupled to a small 1K-word DRAM. All of the DRAMs are addressed in parallel by the row decoder and the distributed column decoders.

We estimate that the Array Chip II-M will contain 600,000 transistors, two thirds of them in DRAM, and will have a die size of 450×600 mils in 1.25-micrometer double-metal CMOS. The target clock rate is 12.5 MHz for a 32-bit, register-to-register ADD over 32 bits (two cells, with one dead cell in the middle). Iterative operations such as multiply will use a higher-frequency clock.

It is important to note that the common bus is 32 bits wide and that the full width of this bus is coupled into each of the 16-bit processors. Furthermore, a

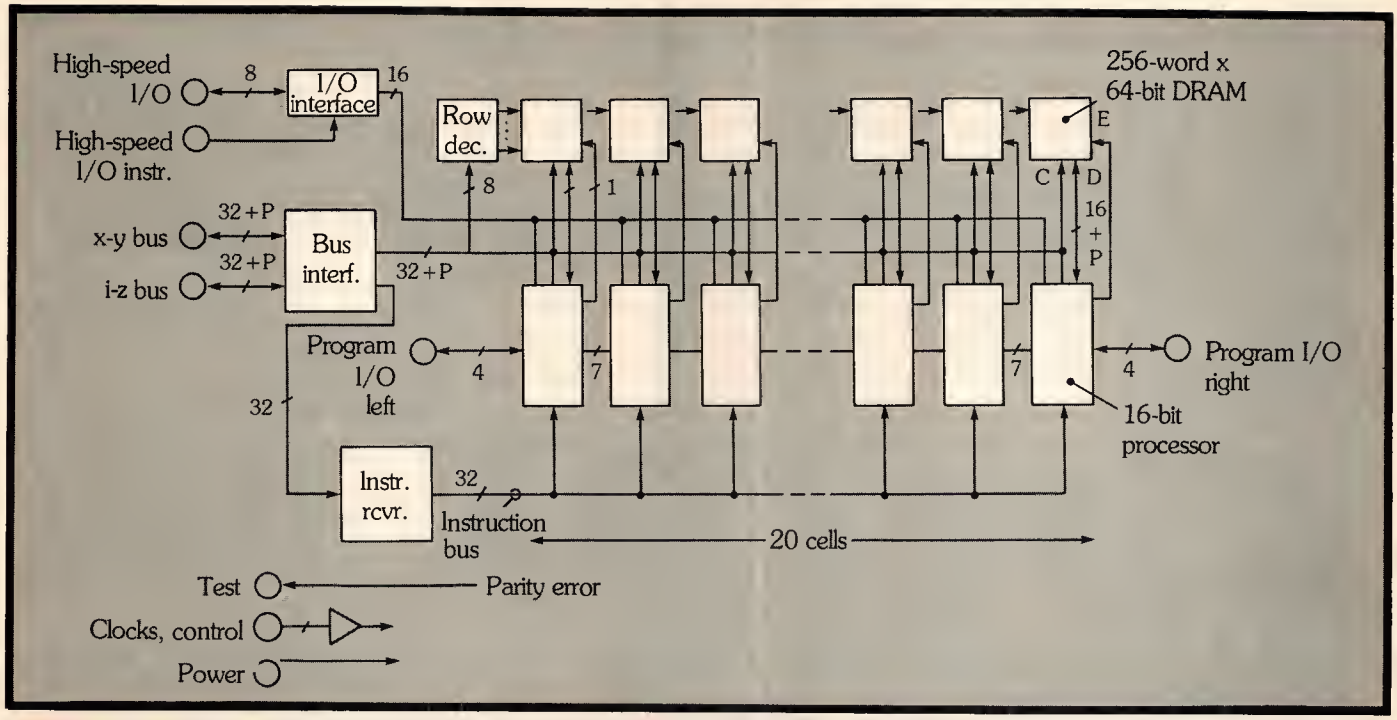


Figure 2. Block diagram of the 256K version of the Array Chip II-M.

16-bit processor, when configured to process a particular portion of a word—e.g., the least significant 16 bits or the most significant 16 bits, is assigned to either the upper half or the lower half of the common bus so that the bus structure will be insensitive to which combinations of 16-bit processors fail.

The address to the local DRAM comes from two 16-bit processors configured to serve as an address generator. The address, which may also be used to reference external vector memory, is passed over the common bus to the row decoder and distributed column decoders, from which it goes to the DRAM. Note that each local DRAM provides a word to the 16-bit processor to which it is connected and that all local DRAMs do so simultaneously, for a net transfer of 256 bits. In a conventional DRAM, only a few bits are passed to a processor, and the many other bits are wasted.

The instruction receiver receives the instruction bus from the external system controller. The instruction bus controls the 16-bit processors. All processors obey the same instruction, or they all ignore it, freezing their state. (We explain this in detail later in this article, in the section on the vector if...else operation.)

Logic cell. The logic cell block diagram for Array Chip II-M is shown in Figure 3. The logic cell is not sensitive to the amount of DRAM on the chip. Thus, improved memory or logic cells can be installed easily as technology evolves.

At the heart of the logic cell is the multiport RAM, or MPR. The MPR provides the working registers, and it stores configuration masks in the processor status word (PSW). The masks are loaded under software control and select whether a cell is active or inactive. If a cell is inactive, it appears invisible and it does not change its state, except possibly to change the configuration. If a cell is active, the mask determines whether the cell is to compute memory addresses (act as a vector address processor) or compute data (act as a vector data processor). The mask also determines how a 16-bit slice is to be used—e.g., as the lower or upper half of the 32-bit address, or as the lower or upper half of a 32-bit data word. Data word sizes of up to 256 bits may be configured.

The MPR accesses two locations: the A address as chosen by the read address (RA), and the B address as chosen by the read/write address (RWA). These two outputs are then operated on as a unit by the ALU according to the instruction, and the result is written back, in the same cycle, into the MPR over the ALU D bus.

The path logic contains the mechanism for hiding defective cells, for connecting cells within a chip to form words longer than 16 bits, and for performing

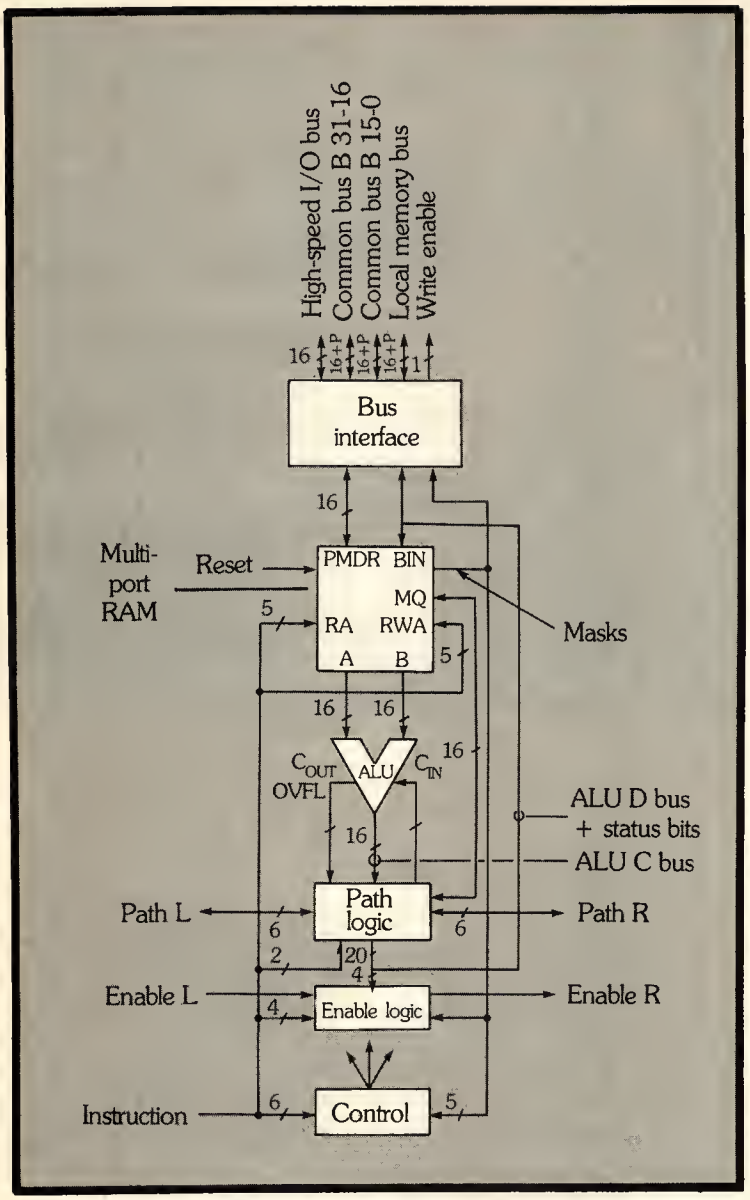


Figure 3. Block diagram of the logic cell for the 256K version of the Array Chip II-M.

functions such as shift and rotate. The path logic of cell n is connected to the path logic on the left cell, cell $n+1$, via the path L (left) signals, and to the path logic on the right cell, cell $n-1$, via the path R (right) signals.

The enable logic contains the physical ID of a cell so that upon system initialization each of the cells can be addressed in turn and its configuration set accordingly. This physical address is 0-19 in the case of a 20-cell chip. After the system has been initialized, a programmer can address a cell by its virtual address, which is stored in the PSW along with the configuration mask.

The control logic decodes an instruction in concert with a configuration mask to determine, for that instruction, how a cell should operate in a particular slice configuration. Numerous multiplexers move the data in whatever way is appropriate to the instruction and the configuration.

The microinstruction set at the array chip level resembles that of a collection of Advanced Micro Devices 2903 four-bit slices. Register-to-register ADD, SUBTRACT, AND, and OR operations are provided, as are iterative multiply, divide, and floating-point operations.

The bus interface provides the connection to the high-speed I/O bus, the upper half and lower half of the common bus, and the local memory. It contains a parity generator and checker that verify the integrity of the various data transfers between a cell and its memory, and between a cell and the outside world.

Programming model

Unlike many array processors that are programmed only by microcode and are used to implement a library of Fortran-callable subroutines, the CAP can run entire applications, often on live data. This is possible because the CAP uses a simple, highly efficient, register-based, load/store architecture with deferred loads and jumps. The design of the instruction set was strongly influenced by the RISC—reduced instruction set computer—philosophy.[4] The set contains only 33 basic instructions, including the load, store, arithmetic, logical, shift, program control, and configuration operations (Table 2). Tables 3 and 4 show the instructions for scientific computation and SIMD support, respectively.

An easy way to program the CAP is to visualize it as a collection of convential SISD processors, each operating out of its own RAM on its own data. The processors operate in lockstep and pass data to the left, right, up, and down to communicate in an organized fashion. Multiple concurrent FFTs or even recursive filters can be readily implemented. (We project that 256 256-point complex FFTs, to 16 bits of precision, will be able to be done concurrently at an average rate of one 256-point FFT every 27 microseconds.)

Figure 4 shows how a 240-pixel × 224-line image can be handled. It is broken into 16 columns. Each column is further broken into 16 rows, for a total of 256 data sets.

Table 2.
Basic instructions.

ADD	Add
ADDC	Add integers with carry bit
ADDI	Integer add immediate
AND	Logical bit-by-bit AND
ASR	Arithmetic shift right (by *n* bits)
BCC	Branch on condition
CALL	Branch to subroutine on condition
CMP	Compare
DIV	Two's-complement integer divide
JMP	Jump on condition
LD	Load MDR from external memory
LSL	Logical shift left (by *n* bits)
LSR	Logical shift right (by *n* bits)
MOVE	Register-to-register move
MUL	Two's-complement integer multiply
NOP	No operation
NOT	One's complement
OR	Logical bit-by-bit OR
RET	Return from subroutine
RTI	Return from interrupt
ROL	Logical rotate left through carry (by one bit)
ROR	Logical rotate right through carry (by one bit)
SAVSYS	Save selected system register
RESSYS	Restore selected system register
STO	Store MDR into external memory
SUB	Subtract
SUBC	Subtract integer with borrow bit
SUBI	Integer subtract immediate
SWI	Software interrupt
TST	Test for zero
TSET	Test and set (semaphore operation)
XOR	Exclusive OR
CONFIG	Configure vector processor for word size and active elements

Figure 5 shows a simplified programming model for the CAP. Each processor has 16 registers, and each vector processor has a local memory. The data processors have processor status words, and the vector data processors have if...else stacks.

Register-to-register operations. Register-to-register instructions use the following encoding format:

Opcode source-reg, destination-reg

The source and destination registers can be chosen from the scalar address register (SAR) group, the scalar data register (SDR) group, the vector address register (VAR) group, or the vector data register (VDR) group. For each source and each destination, two bits select one of these four register groups, and four bits select a register within that group.

"Register 0" in every register set is designated as the memory data register, or MDR. All memory accesses must originate from, or be destined for, this register. In fact, the load and store instructions imply the MDR as an operand.

A parallel operation is specified by a vector-type source and/or destination, i.e., by a VAR or VDR, rather than by an instruction that specifies a vector operation as a part of its mnemonic. A complete, uniform set of operations is thus ensured and, depending on the operand register type, different kinds of transfers can occur. Transfers can be either single

Table 3.
Scientific computation instructions.

ADDF	Floating-point add
DIVF	Floating-point divide
MULF	Floating-point multiply
SUBF	Floating-point subtract
SQRTF	Floating-point square root
FIX	Convert a floating-point number into integer format
FLOAT	Convert an integer into floating-point format

Table 4.
SIMD support instructions.

PUSHIF	Push test condition on VIE stack
COMIF	Complement top of VIE stack
POPIF	Pop VIE stack
FIND	Find and enable the first active vector processor (special PUSHIF)
DROP	Disable the vector processor selected by FIND (special COMIF)
UP	Move MDR in each processor to MDR of processor above
DOWN	Move MDR in each processor to MDR of processor below
LEFT	Move MDR in each processor to MDR of left processor
RIGHT	Move MDR in each processor to MDR of right processor

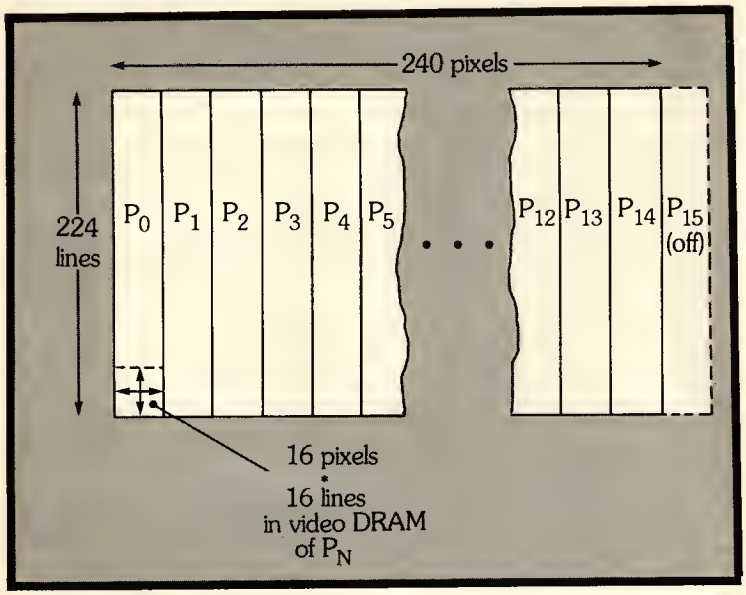


Figure 4. Image partitioning on the CAP.

register move (1:1 or 1-of-many:1), register broadcast (1:many), or vector registers move (many 1:1). The register combination indicates the type of transfer that is to take place.

The example in Table 5 illustrates how the move instruction is interpreted.

Addressing modes. Only the load and store instructions can access the memory. There are five different addressing modes used to calculate the effective address of the operand. These modes, which also apply to the call and jump instructions, are used to calculate the effective jump address. They are

(R)	Register indirect
(R)++	Register indirect, postincrement
--(R)	Register indirect, predecrement
(R+offset)	Register plus 32-bit offset, indirect
#ABS	Immediate absolute address

We support the vector processor memory hierarchy by providing three sets of load and store instructions—one for the local vector memory, one for the vertical vector memory, and one for the horizontal vector memory. We also provide block roll-in and roll-out operations.

Conditional vector processor execution. In a conventional SISD system, conditional execution can be easily expressed with high-level if...else statements. It is very clear how the following pseudocode fragment works in a SISD system:

```
for i = 1 to 8 do
  if ( A[i] > B[i] ) then_begin
    statement C;
  else_begin
    statement D;
  end_if;
end_for;
```

During program execution, the system tests the value of A[i] against B[i] for all i, one after another. If the variable A[i] is greater than the variable B[i], then statement C is executed; otherwise, statement D is executed. At first glance, we thought the if...else programming construct could be easily adapted to SIMD

Table 5.
Interpretation of the move instruction.

	Source	Dest.	Operation
1.	SAR*x*	SAR*y*	1:1
2.	SAR*x*	SDR*y*	1:1
3.	SAR*x*	VAR*y*	1:many—broadcast SAR*x* to all VAR*y*'s
4.	SAR*x*	VDR*y*	1:many—broadcast SAR*x* to all VDR*y*'s
5.	SDR*x*	SAR*y*	1:1
6.	SDR*x*	SDR*y*	1:1
7.	SDR*x*	VAR*y*	1:many—broadcast SDR*x* to all VAR*y*'s
8.	SDR*x*	VDR*y*	1:many—broadcast SDR*x* to all VDR*y*'s
9.	VAR*x*	SAR*y*	1-of-many:1—select by a find instruction
10.	VAR*x*	SDR*y*	1-of-many:1—select by a find instruction
11.	VAR*x*	VAR*y*	many 1:1—in each active vector address processor
12.	VAR*x*	VDR*y*	many 1:1—in each active vector processor
13.	VDR*x*	SAR*y*	1-of-many:1—select by a find instruction
14.	VDR*x*	SDR*y*	1-of-many:1—select by a find instruction
15.	VDR*x*	VAR*y*	many 1:1—in each active vector processor
16.	VDR*x*	VDR*y*	many 1:1—in each active vector data processor

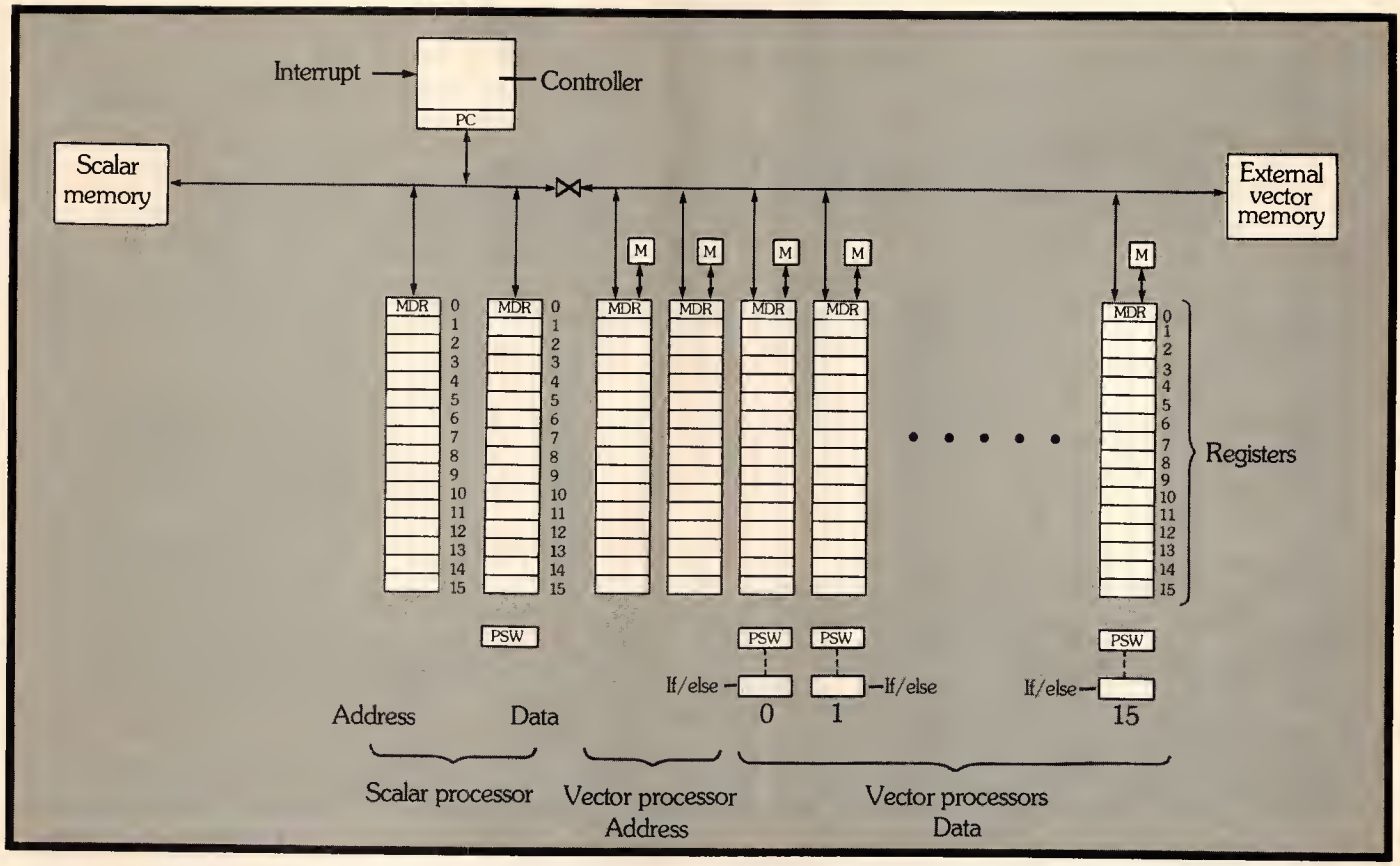


Figure 5. Simplified programming model for the CAP.

programs (i.e., to the vector mode). However, the hardware implementation of the vector if...else operation is more difficult than it appears. This is true because the control flow of the if...else operation is intrinsically scalar and data-dependent: If the data has a certain value, the processor executes only a small part of the program.

On the other hand, the SIMD architecture requires all the vector processors to receive the same instruction simultaneously. However, since each processor presumably has its own unique data, then only some of the processors should execute the instruction stream.

Consider the operations when the pseudocode presented above is translated into a vector format:

```
for each_processor, P[i], i = 1 to 8, do
   if ( A[i] > B[i] ) then_begin
      statement C;
   else_begin
      statement D;
   end_if;
end_for;
```

During execution, each vector processor P[i] tests the value of its A[i] and B[i]. All tests occur simultaneously. Some may satisfy and others may fail the test condition. Those processors that satisfy the condition have to execute statement C but not statement D, and those that do not have to execute statement D but not statement C. The question now becomes how to satisfy these two cases when all the processors in a SIMD system receive the same instruction stream simultaneously.

The instructions for all cases must be transmitted. The CAP provides three instructions to support the vector if...else concept:*

- The PUSHIF COND instruction is a primitive for the if statement. It evaluates the specified conditions (=0, > 0, etc.) and pushes the single-bit result on the vector if...else stack in each vector data processor. A vector processor is active if the stack is empty or all pushed bits are true. Otherwise, it is disabled.
- The COMIF instruction is a primitive for the else_begin statement. This instruction complements the top of the vector if...else stack, performing the func-

*Note that in this discussion the term "instruction" means a CAP assembly-level instruction, whereas the term "statement" means a high-level-language statement.

tion of the else_begin statement. A vector processor with a single false condition on the top of the stack thus returns to an active state.

- The POPIF instruction is a primitive for the end_if statement. It pops the vector if...else stack, undoing the effect of the last PUSHIF or COMIF instruction.

We feel that for many classes of problems this method of vectorizing the if...else structure provides a good alternative to conventional methods. Furthermore, nested vector if...else structures can easily and efficiently be achieved by cascading these instructions.

Project history and status

In 1982 we rapidly roughed out a highly parallel architecture that employed 1024 one-bit processing elements in a 32×32 x-y array. Unlike the processors in many cellular SIMD designs, these one-bit elements could operate on words in either bit-serial or bit-parallel fashion, as selected by software.

We planned to put 16 of these one-bit processors on a single chip that was to be fabricated in four-micrometer double-metal CMOS. The projected size of this chip was enormous, nearly six tenths of an inch square. Not only would the yield of such a chip be near zero, but the layout job appeared to be monumental. Few people expected the project to survive.

We then made perhaps our most significant breakthrough. We reasoned that chip fabrication defects are a fact of life and that we should learn to live with them. We realized that the highly regular design of our chip, in which the identical cells can be viewed as interchangeable parts and in which the bit significance of a cell can be programmed by software at run time, provided the basis for fault tolerance. (We call our approach "dynamic fault tolerance" because of the run-time software control.) We simply provided enough spare cells to compensate for the average number of cells that we expected to be defective at the time of manufacture. Furthermore, we could use any remaining spare cells as replacements for cells failing during the life of the system.

This technique enabled us to use "large-area integration." We can now build chips that are large enough to expect and deal with defects but small enough to minimize catastrophic defects. We can economically put far more memory and logic on a single chip than would otherwise be possible. System performance and reliability improve, while size and cost decrease.

Because of the complexity and size of the chip, the CAP team, which was involved in both the architectural and VLSI design, had to develop its own CAD tools as well as integrate commercially available tools with them. Behavioral modeling concepts were used to verify the architecture of the design. A Zycad logic simulator engine[5] was used in conjunction with extraction and layout analysis programs written by the team to verify and check the detailed design. Logic synthesis programs were also written to aid the design and optimization of PLA control equations.

We did the VLSI layout using two tools that we developed. We extended the Berkeley "cabbage" program for symbolic layout by making it hierarchical, creating the "hierarchical cabbage" program.[6] A chip of any size can in principle be built with this tool. In addition, we wrote a procedural layout language which we used to generate the PLAs and the pad buffers.

During the time we were designing the CAP, fabrication technology improved to three-micrometer double-metal CMOS. We laid out our first chip—Array Chip I—in 1984.

Array Chip I. This device is our first-generation processor chip and is the basis for the prototype CAP system we are currently building. It demonstrates the basic fault-tolerant techniques upon which the entire CAP system design is dependent. It contains 20 one-bit cells, of which only 16 are required for proper system operation. The locations of bad cells are invisible to the I/O pins thanks to an on-board, software-controlled, cellular switching matrix.

Array Chip I may be viewed as a 16-bit slice, although any combination of word sizes may be selected by software, with no external logic required. The chip has 120,000 transistors, 144 pins (many no longer used), a power consumption of about 500 mW, and a die size of 650 × 500 mils. Its target clock rate for a 16-bit register-to-register add over 20 cells is 10 MHz, although its actual clock rate is closer to 5 MHz.

A photograph of the chip is shown in Figure 6. One can readily see the highly regular structure which greatly simplified the chip's design. There are four groups of five one-bit cells each, with the groups separated by power and ground runs. Pin drivers and receivers and a small amount of common logic surround the cells. A large amount of PLA per cell (the dense, dark bands) was required; it, among other factors, resulted in a relatively low computation rate per transistor.

Feasibility model. We will use Array Chip I in the prototype CAP we are now building to demonstrate

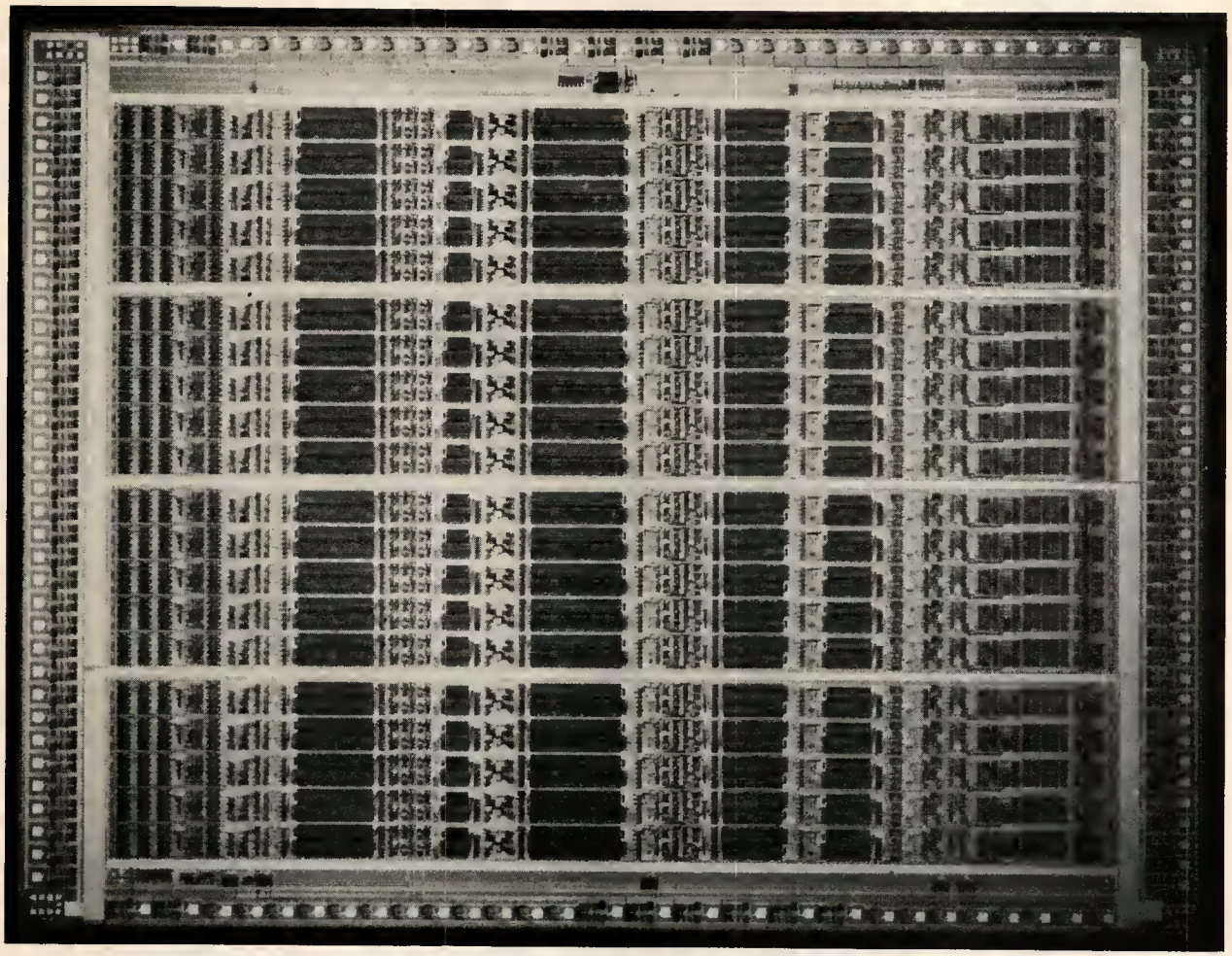


Figure 6. ITT CAP Array Chip I, Revision 0. This 120,000-transistor device is 650 × 500 mils and has a power consumption of about 500 mW.

system feasibility. Its use will show that a high-performance parallel processor can be built out of imperfect parts. The machine will combine 36 Array Chip I's with conventional TTL control logic to provide a peak performance of about 100 MIPS for 16-bit words. It will execute most of the instruction set described in this article.

The CAP system block diagram is shown in Figure 7. The scalar execution unit contains 32-bit paths, whereas the vector execution unit contains 16 vector processors with 16-bit paths. The programming model for the prototype CAP is very similar to the model we described earlier. The basic difference is that in the prototype the data widths will be fixed at 32 bits for scalar data and at 16 bits for vector data. The hardware will handle the truncation or sign extension required for data transfers between the vector and scalar processors.

Our particular application is in image processing, which requires a high data transfer rate. Hence, we are providing the prototype with an I/O bus that will be separate from the main memory bus. A FIFO in each vector processor will store a block of words from an I/O device, and when all the rows have a block of words for loading into vector memory, an interrupt will be generated. An interrupt service routine will then cause each of the rows to transfer data from its FIFO to its vector memory, from which the image processing will then be performed. The vector memory in our prototype will be made up of video DRAMs to provide data output to a video display. All application coding is currently in assembly language, but we are planning to develop at least one high-level language in 1986.

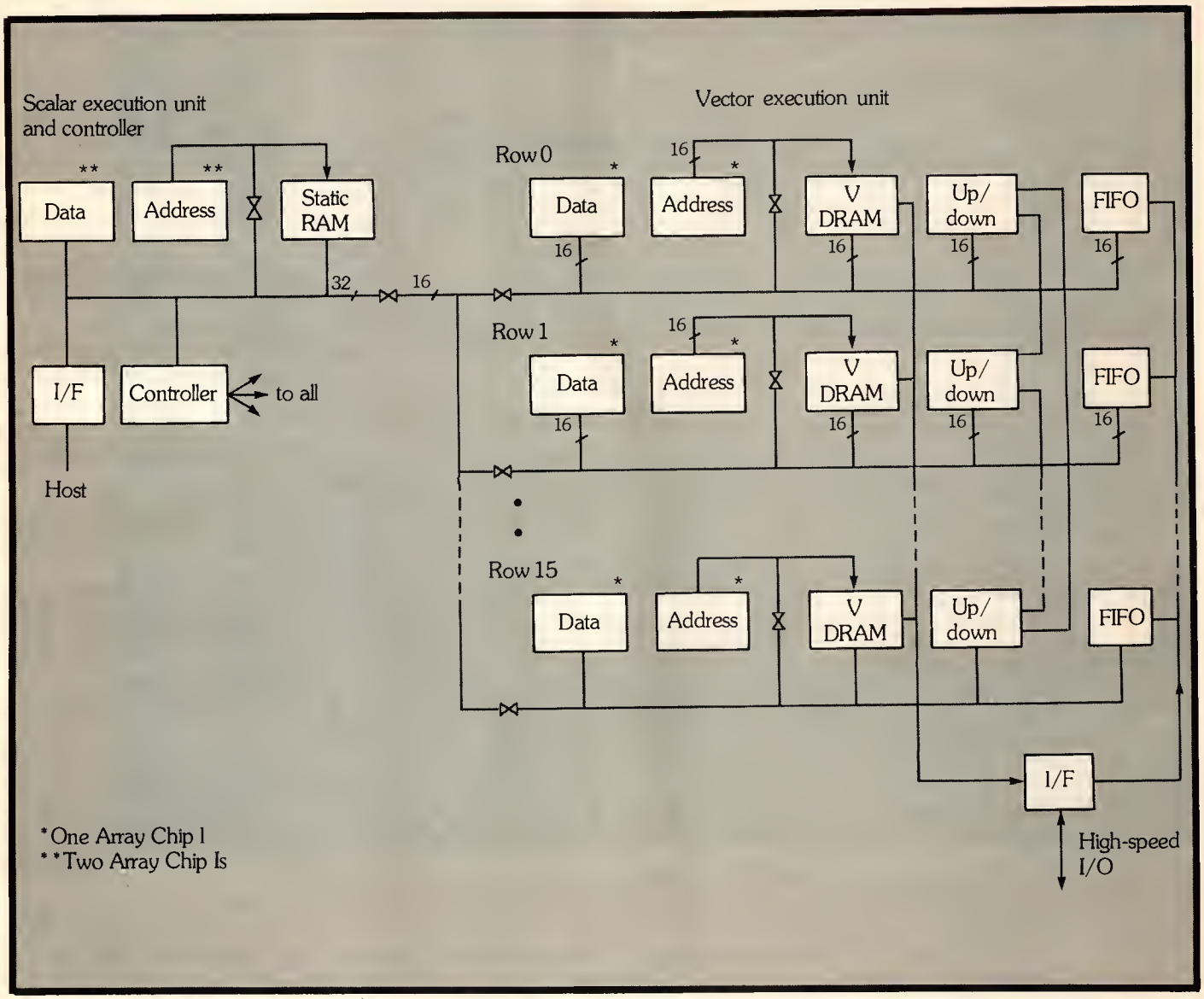


Figure 7. Prototype ITT CAP architecture.

The ITT Cellular Array Processor's ability to tolerate faults and its highly regular, memory-intensive, bit-parallel architecture promise to make it much more powerful, and yet smaller and less expensive, than competing designs. Such unprecedented capability can be expected to provide dramatic benefits in signal and image processing, in engineering analysis, and in database manipulation.

Can the age of the personal supercomputer be very distant if the CAP project achieves its objectives?

Acknowledgments

The authors thank Jack Cotton of the Systems Research Division of the ITT Advanced Technology Center for initiating the CAP Project. His belief in parallel processors and in the team's ability to conquer what seemed to be overwhelming obstacles paved the way for the design described here.

The authors also thank the staff of *IEEE Micro* for enabling this article to be published on a very short timetable.

References

1. Chuan-lin Wu, guest ed., Special Issue on Multiprocessing Technology, *Computer*, Vol. 18, No. 6, June 1985, pp. 6-108.
2. Eric J. Lerner, "Parallel Processing Gets Down to Business," *High Technology*, Vol. 5, No. 7, July 1985, pp. 20-28.
3. Kenneth E. Batcher, "Design of a Massively Parallel Processor," in *Tutorial on Parallel Processing*, Robert H. Kuhn and David A. Padua, eds., IEEE Computer Society Press, 1981, pp. 80-85.
4. David A. Patterson, "Reduced Instruction Set Computers," *Comm. ACM*, Vol. 28, No. 1, Jan. 1985, pp. 8-21.
5. D. Jenkins and S. Morton, "Transistor-Level Logic Simulation Using the Zycad Logic Evaluator," *Proc. ADEE East Conf.*, Oct. 1985, pp. 154-163.
6. C. Barrila, D. Jenkins, E. Abreu, and S. Morton, "Hierarchical Cabbage Symbolic VLSI Layout Tool," in preparation.

Steven G. Morton has been the principal architect of the ITT Cellular Array Processor since 1982. He joined ITT in 1979, and his first responsibility was the construction of a signal processor for the ITT System 12 Digital Switch. Prior to ITT, he was with the MIT Lincoln Laboratory, where he worked on attitude control and ground-based telemetry systems for the LES-8 and LES-9 Air Force communications satellites. He received his BSEE and MSEE from MIT in 1972.

Enrique Abreu is the lead VLSI designer for the ITT CAP Project. He joined the project in 1982. He received his BS in mechanical engineering and electrical engineering from the University of the Philippines in 1980 and his MSEE from the University of Pennsylvania in 1982.

Fred Tse is responsible for the detailed design of the CAP instruction set and CAP controller. Prior to joining ITT in 1984, he worked for three years at AiResearch Corporation, where he designed real-time digital controllers, and for five years at Bell Labs in Denver, Colorado, where he designed systems for system test applications. He received his BSEE from Mississippi State University and his MSEE from the University of Southern California.

Questions about this article can be directed to Steven G. Morton at the Applied Technology Division of the ITT Advanced Technology Center, 1 Research Drive, Shelton, CT 06484; (203) 926-5739.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 165 Medium 166 Low 167

A Performance Analysis of MC68020-based Systems

Doug MacGregor
Motorola Inc.

Jon Rubinstein
Hewlett-Packard

The determination of processor performance during the early stages of design is essential to the proper analysis of design trade-offs as well as to the evaluation of product viability. This problem is common to both the developer of the processor and the developers of systems based on the processor. In each case, an understanding of the processor's behavior is essential to proper design.

Here, we describe a method for estimating the performance of the MC68020, a 32-bit microprocessor, and computer systems based on the 68020. We undertook two separate studies in developing this methodology. First, we devised a general model of computation for the 68000 by tracing typical programs executing on a 68000-based workstation. From these traces, we developed a profile of instruction frequency that served as a model for processor execution. Second, we made traces of bus activity in a 68000-based Hewlett-Packard 9000, Series 200, Model 236 computer. We used these traces to develop a model of bus behavior—a model that includes locality, types of accesses, and DMA activity. (This model was later used in the design of the new HP 9000, Series 300 computer family.)

By combining these models, we can predict the performance characteristics of the 68020 microprocessor and any system based on the 68020. In this article, we will examine the models to understand the factors affecting performance of the processor and a system based on the processor. We will then correlate the predicted performance characteristics to the 68020 chip and system, respectively, to determine the validity of our method.

When developing processor-specific computation models that will be used to evaluate performance, determine resource utilization, and describe the nature of processing, one should avoid comparing the results to those produced by a different architecture. However, the *methods* we used to develop the models of computation and bus behavior are applicable to, and valuable for, the evaluation of any architecture.

Performance measurement. One of the difficulties inherent in attempting to measure performance is that performance depends to a large degree on the application and the environment of the computer or microprocessor.[1] It is even difficult to find units of measurement that are not controversial. If one uses execution times—which provide an absolute reference—the characteristics of the system can overshadow the performance of the processor. As a result, the use of relative measurements such as MIPS (millions of instructions per second) is common. Use of these relative measures enables us to eliminate other factors and describe the capabilities of the processor alone.

Relative measures do not easily support the comparison of different processors. It is wrong to compare the MIPS rating of one architecture to that of

0272-1732/85/1200-0050$01.00 © 1985 IEEE

another, just as it is to compare the MIPS rating of a processor to that of an entire system. In each case, there are too many variables that the simplistic MIPS rating, by design, cannot account for. If one wishes to compare MIPS values across architectures, one must develop a factor that accounts for architectural differences and that can be used to adjust those MIPS values. It is this factor which causes controversy in the comparison of computer systems.

In choosing a microprocessor, one must consider its performance both in an optimal environment, where it never waits for an external resource, and in a system environment, where it frequently waits for an external resource. However, measurements made in one environment must never be compared to those made in the other, since the complexity of the system environment degrades the potential performance of the processor.

Throughout this article, measured performance is reported in absolute times and in MIPS values standardized to the 68000 architecture. At present, we cannot determine a reliable factor for adjusting MIPS between the 68000 and the 68020, due to a lack of 68020-specific compilers. However, we made simple adjustments to a 68000 compiler that resulted in factors ranging from 1.00 to 1.14 for the benchmarked programs. To be conservative, we used a factor of 1 and thus treated the two architectures equally, even though the 68020 is a superset of the 68000.

The 68020. The 68020 microprocessor maintains compatibility with the Motorola 68000 family while providing several architectural enhancements: full 32-bit operations (by extending the few instructions that were not already 32-bit), floating-point operations (through the MC68881 floating-point unit), high-level-language programming (with support for scaled indexing), graphics and communication processing (with bit-field instructions), and support of any type of user- or Motorola-defined coprocessor (through the coprocessor interface).

The architecture of the 68020 was established by the 68000 and the 68010.[2,3] Its virtual memory and virtual machine support mechanisms are the same as those used in the 68010.[4] The 68020 uses the flexible, nonmultiplexed, asynchronous 68000 bus, extended to 32-bit addressing. It performs dynamic bus sizing so that elements of the system can be connected on an 8-, 16-, or 32-bit data bus. A complete description of the 68020 can be found in MacGregor[5] and in the user's manual.[6]

68020 performance. Enhancements accounting for the improved performance of the 68020 over the 68000, and the contribution of each enhancement to that performance, are shown in Table 1. The 68020 and an 8-MHz 68000 were compared, with no external constraints on either processor. Although the 68020 is a 16-MHz device, the comparison was valid, since the 68020 was designed to operate at a worst-case clock frequency of 16.67 MHz and the 68000 and 68010 were designed for worst-case clock frequencies of 8 MHz. Faster 68000s and 68010s are now sold, and in the future faster 68020s can be expected.

Table 1.
Factors affecting performance.

68020 enhancement	Performance factor
Higher clock frequency	2.00
32-bit data bus	1.35
Instruction cache	1.10
Three-clock bus cycles	1.15
Internal enhancements	1.08
New instructions and addressing modes	1.05
Instruction overlap	1.05
Total	4.07

A large portion of the 68020 performance improvement is attributable to the higher clock frequency. The obvious benefit of increasing the clock frequency is that all internal resources are correspondingly faster. However, the performance improvement provided by the higher clock frequency depends heavily on system access time.

System access time is the most important external factor directly affecting performance. To minimize access time, the 68020 incorporates features designed to reduce its dependence on memory speed. These include the 32-bit data bus, the instruction cache, and three-clock bus cycles.

Among other features contributing to the higher performance are new instructions and addressing modes, a barrel shifter, and special multiply logic. The degree to which each of these contributes depends on the application. Interinstruction parallelism was introduced in the 68020 in the form of instruction overlap. Overlap occurs when a data write or an instruction prefetch takes place in parallel with the execution of the next one or more instructions. Overlap is enhanced by use of the instruction cache, which reduces bus utilization and, thus, resource competition.

68020 system configurations. A major factor affecting the performance of a computer system is the raw power of the processor. However, the processor is still just one element of the system, and one must characterize the entire system in order to evaluate its potential performance. However, because so many 68020-based system configurations are possible, one cannot characterize all conceivable systems. Therefore, we will discuss only a reasonable subset of the possible implementations.

68020 system configurations can be divided into three categories: single-board computers, plug-in upgrades, and new system designs. At the low end of the price range are single-board implementations. Because of the instruction cache and the 32-bit data bus, these can provide exceptional performance. If the design goal is to build a system quickly, the 68020 can easily replace a 68008, 68000, or 68010 in an existing implementation.[7,8] If full utilization of the 68020's capabilities is the goal, however, a completely new system must be designed.

The factor under direct control of the hardware designer that most affects system performance is the processor's access time to memory. Because both the processor and the memory subsystem have such a significant impact on a system's performance and price, they must be well matched for the system to operate efficiently.

In a typical system, the 68020 can access memory at a rate higher than the rate most memory subsystems can handle. The hardware designer could solve this by using faster, more expensive RAMs in the memory subsystem. A less expensive solution is possible, however—a high-speed cache memory can be placed between the processor and main memory. A cache isolates the processor from the slower main memory subsystem. The design of the cache will dictate both the access time of the processor and its bus utilization. A complete discussion of cache design is provided by Smith.[9]

In developing a new system, one must maintain compatibility with older members of the family. The time required to port software from an old system to a new, incompatible one is becoming astronomical and is already dominating the overall system development cycle. But if the old system is based on the 68000, for example, and a compatible processor such as the 68020 is used in the new system, the development time is significantly reduced. Maintaining compatibility also allows the computer vendor to easily upgrade customer equipment and prevents customers from becoming "computer orphans."

Software compatibility in the 68000 processor line is demonstrated by the computers in the HP 9000 Series 200 and 300 family. Though Series 200 models use the 68000 and lower-performance versions of the 68010, and Series 300 models use the 68020 and higher-performance versions of the 68010, every member of the family is user software-compatible with every other member, allowing easy upgrading from one model to the next.

Performance of the processor

Here, we will consider the performance of the processor alone. Using instruction trace data representing dynamic instruction frequency, we present a model of computation for a 68000 operating in a general processing environment. From this model, we obtain an estimate of the performance and bus utilization of both the 68000 and the 68020. We also identify and analyze the design elements that contribute to the 68020's performance. Finally, we correlate the estimated performance of the 68020 to its actual performance in a variety of system configurations.

We made our performance measurements on general-purpose HLL (high-level-language) applications executing under a UNIX-like operating system. We selected 20 programs from four categories: traditional benchmarks, operating system utilities, program development tools, and application programs originally used in the 68020 development. We compiled these programs using five different compilers* and executed them on the 68000 to determine the dynamic frequency** of execution for the 68000 instruction set. These programs form a composite of over 500 million lines of executable machine code.

We estimated the execution time and performance characteristics of the 68020 by appropriately weighting the instruction execution times after taking into account the system access time. Our evaluations do not consider sequence-dependent factors such as instruction overlap, in which instructions are executed in parallel. As a result, our performance estimates tend to be conservative and indicate a slightly lower performance level than is actually obtained.

*Because HLL programs must be compiled, there is a dependence on the compiler used. While it would be interesting to perform an evaluation on the HLL source programs to remove the influence of the compiler, this approach has two problems: the difficulty of correlating HLL statements to an architecture, and the difficulty of performing dynamic analysis. While studies examining the nature of languages have been performed, they quickly lose their meaning when related to a particular architecture.[10]

**Until the relationship between static and dynamic frequency is affirmed for an architecture, the use of dynamic frequency of machine instructions is preferred.[11]

The nature of general processing. Understanding the application being executed is fundamental to any discussion of performance. The application selected for our study is a composite of 68000 programs written in high-level structured languages and operating under a UNIX-like operating system. It is a profile of a typical engineering workstation. (Because floating-point instructions are not supported by the 68000, this important class of instructions is not represented in our figures. It will not be possible to analyze floating-point behavior until data can be collected from a 68020/68881 system.)

Table 2 shows basic 68000 instructions by type, along with their respective dynamic frequency and

Table 2.
Instruction frequency and execution time.

General class	Dynamic frequency	68000 dynamic exec. time	68020 dynamic exec. time
MOVE	32.85%	36.86%	32.41%
Move reg, reg	2.98%	0.97%	0.95%
Move reg, mem	6.39%	6.20%	4.91%
Move mem, reg	15.84%	17.96%	17.46%
Move mem, mem	7.20%	11.59%	8.65%
Move (other)	0.44%	0.14%	0.44%
BRANCH/JMP	23.95%	18.73%	26.87%
Branch cond	15.03%	11.42%	15.41%
Decr and branch	8.37%	6.81%	10.74%
Jump	0.55%	0.50%	0.72%
CMP/TEST	10.95%	10.53%	11.54%
Compare	7.86%	7.61%	8.17%
Test	3.09%	2.92%	3.37%
ARITH/LOGICAL	16.31%	17.12%	13.76%
Arithmetic/logical	11.47%	9.20%	7.35%
Multiply	0.58%	2.90%	2.69%
Divide	0.13%	1.54%	1.04%
Shift and rotate	3.25%	3.19%	2.12%
Extend	0.88%	0.29%	0.56%
SUBROUTINE	6.14%	7.72%	7.15%
Branch subroutine	0.98%	1.44%	1.42%
Jump subroutine	0.71%	0.93%	1.03%
Return subroutine	1.69%	2.20%	3.26%
Link and unlink	2.76%	3.15%	1.44%
MISCELLANEOUS	7.53%	9.02%	7.26%
Bit operation	0.22%	0.18%	0.14%
Clear	2.69%	3.93%	2.50%
Load effective addr	2.40%	2.11%	2.08%
Push effective addr	1.85%	2.41%	2.08%
Set conditional	0.37%	0.39%	0.46%
TOTAL	97.73%	99.98%	98.99%

Note: Dynamic execution times are based on the percentage of time for the instructions counted in the dynamic frequency category. In this case, the 2.25-percent dynamic frequency not attributed is averaged into the execution times.

dynamic execution times for the 68000 and the 68020. The dynamic frequency of execution shows how often an instruction is executed, whereas the dynamic execution times reflect the percentage of the total execution time each of the processors spends executing an instruction. The table also summarizes dynamic frequency and dynamic execution times by instruction type. Execution time depends on external as well as internal factors. The figures shown are based on no-wait-state memory accesses and on an instruction cache hit rate of 64 percent for the 68020.

Hit rates for an instruction cache can be expressed either as a percentage of instruction accesses or as a percentage of all accesses. Because 68000 instruction words are 16-bit, the cache hit rate can also be considered for each 16-bit instruction word needed to fill the pipe or for each 32-bit long word fetched. The method selected to measure hit rates has no effect on performance measures as long as the same units are used throughout the study, or the appropriate adjustments are made. We collected the data on cache performance from 68000 traces, which fetch 16 bits at a time. In order to use this data with minimal modification, we described the hit rate as the percentage of 16-bit instruction requests that resides in the instruction cache. Using this definition, we observed the aforementioned hit rate of 64 percent for the 256-byte 68020 instruction cache. We will discuss how we determined this value in greater detail later.

From the dynamic execution times we can determine which mechanisms in the processor are slower than others. It is important to note that the percentage of execution time is a relative measure—it is meaningful within the processor but has no relationship to any other processor or to the physical execution time. There are no instructions that are slower, in absolute terms, on the 68020 than on the 68000, but for every instruction made made proportionally faster on the 68020, there must be one proportionally slower. For example, if the move instruction took no time, the processor would be much faster, but all other instructions would take a larger portion of the execution time, even though their absolute execution times would be the same.

The way in which the processor deals with memory is critical to performance. Table 3 indicates how the operand or operands of an instruction are accessed. These operands can be immediate, memory, or register values; they are stored in the instruction stream or, possibly, in the cache, memory, and internal register set, respectively. The first column of Table 3 shows the dynamic frequencies obtained in source EA (effective addressing) modes.*

The most common addressing mode for the source operand of a dyadic operation is register-indirect with displacement, which represents the read of an HLL argument. Typically, a base pointer to the argument area from which an operand is accessed is maintained by providing a displacement off the base pointer corresponding to that operand. Table 3 does not reflect the new addressing modes that have been added to the 68020 to improve the efficiency of array accessing.

The second column in Table 3 shows the dynamic frequencies obtained in destination EA modes. Because the 68000 architecture does not support self-modifying code, the possible destination addressing modes are reduced to register or memory locations—they cannot utilize the instruction stream. This simplifies the system interface to the instruction cache tremendously. The most commonly used destination addressing mode is register-direct. Because the access time of an on-chip register is much faster than that of memory, a compiler should use registers as often as possible and only use memory accesses to fetch the operand for the first time and to store the results of the completed operation.

Profile of an average instruction. From the dynamic instruction frequency data we developed a profile of the average executed instruction. Table 4 shows the average number of clocks in an instruction for the 68000, 68010, and 68020. Because of the instruction cache, three values for the 68020 are provided—one with the cache disabled, one with the cache hitting 64 percent of the instruction accesses, and one with the cache hitting 100 percent of the instruction accesses.

From this table, we can see the performance improvement of the 68020 over its predecessors. The relationship between MIPS and the average number of clocks per instruction is direct and is given by the following equation:

$$\text{MIPS (instr/sec)} = \text{CR (clocks/sec)} / \text{CPI (clocks/instr)}, \quad \textbf{(Equation 1)}$$

*There are a number of instructions which do not have a source operand, including those instructions which have no operands (such as branch) as well as the monadic instructions. Most source EAs relate to the fetching of operands, although in some cases (6 percent) the EA itself is an operand (e.g., load effective address). In all other cases, the source EA modes refer to dyadic operations.

where CPI = the total clocks per instruction (see again Table 4) and CR = the clock rate in MHz. The lower number of average clocks per instruction reflects the performance enhancements made to the processor in addition to the increase in the basic clock frequency. From Table 4 we can determine the performance in MIPS for a processor executing with a no-wait-state memory. This performance is shown in Table 5 for a variety of clock frequencies.

Table 6 shows the average number of accesses per instruction by access type. By using the figures in this table with those in Table 4, we can estimate the performance of a processor operating with a delay—i.e., we can estimate the average number of clocks per in-

Table 3.
Source/destination addressing mode frequency.

General class	Source EA frequency	Destination EA frequency
NONE	35.10%	17.28%
REGISTER-DIRECT	12.50%	49.48%
Dn	2.15%	24.30%
An	1.47%	7.56%
Rn	8.88%	17.62%
REGISTER-INDIRECT	30.71%	25.44%
(An)	4.08%	1.50%
(An)+/−(An)	4.16%	13.00%
(disp + An)	20.10%	9.13%
(disp + An + Xn)	2.37%	1.81%
ABSOLUTE	0.54%	0.00%
XXX.W	0.14%	0.00%
XXX.L	0.40%	0.00%
PROGRAM-RELATIVE	0.42%	—
(disp + PC)	0.32%	—
(disp + PC + Xn)	0.10%	—
IMMEDIATE	7.77%	—
#XXX	6.57%	—
Quick	1.20%	—
IMPLIED STACK	3.08%	4.93%
TOTAL	90.12%	97.13%

Table 4.
Average number of clocks per instruction.

	68000	68010	68020 0% hit	68020 64% hit	68020 100% hit
0 wait states	12.576	12.107	7.682	7.159	6.373

Note: To determine the effect of wait states on the average number of clocks per instruction, add the total number of accesses per instruction (see Table 6) times the number of wait states.

struction for a processor operating with a memory using any number of wait states:

TCPI (clocks/instr) = BCPI (clocks/instr) + (AAPI (accesses/instr) * NWS (clocks/access))
MIPS (instr/sec) = CR (clocks/sec) / TCPI (clocks/instr) **(Equation 2)**

where AAPI = the average number of accesses per instruction (from Table 6), BCPI = the basic clock rate per instruction (from Table 4), CR = the clock rate (in MHz), NWS = the number of wait states per access, and TCPI = the total clocks per instruction. This approach is a bit simplistic and will slightly underestimate performance, since in many cases the first wait state will not add a clock delay. However, it is very accurate for cases involving more than one wait state and yields a reasonable estimate. We prefer this approach because of the inherent inaccuracy of performance estimates which do not take into account complex relationships that are sequence-dependent, and because of our desire to have a conservative number.

As system access time increases and as the number of wait states increases, eventually the only factor determining performance is the number of accesses made. It is useful to develop a better understanding of these accesses since they exert such an influence on total processor performance. In Table 6, the average number of accesses per instruction is shown for the 68000, 68010, and the 68020. The 68020 has values for both bounds of the cache as well as for the expected 64-percent hit rate. The accesses are broken down into types to show the average number of accesses per instruction for instruction fetches, data reads, and data writes. This information is of value to those building an external (off-chip) cache, who must develop a strategy to deal with problems like stale data associated with writes and the size of the cache given the locality and hit rates of various types of access. From the differences between the number of accesses made by the 68010 and the number made by the 68020, we can determine the number of long data reads and writes. Since the basic number of operand accesses are constant for a program, any decrease from the 68010 to the 68020 in esti-

Table 5.
Performance in MIPS for a processor with no-wait-state memory.

	68000	68010	68020 0% hit	68020 64% hit	68020 100% hit
8 MHz	0.64	0.66	—	—	—
10 MHz	0.80	0.83	—	—	—
12.5 MHz	—	1.03	1.63	1.74	1.96
16.6 MHz	—	—	2.17	2.33	2.62

Table 6.
Average number of accesses per instruction.

Access type	68000	68010	68020 0% hit	68020 64% hit	68020 100% hit
Instruction fetch	1.662	1.662	0.958	0.575	0
Data read	0.644	0.602	0.384	0.384	0.384
Data write	0.392	0.392	0.242	0.242	0.242
Total	2.698	2.656	1.584	1.201	0.626

Table 7.
Percent of accesses that are long.

	Extracted	Simulated
Data reads	57%	54%
Data writes	62%	58%

mated bus activity is due to the increased bus width of the latter.*

We also simulated the performance of the processor on a 16-bit bus. Employing a method like the one above, we used the results of this simulation to determine the percentage of long-operand transfers. The percentage of accesses that are long is shown, by type, for the 68020 in Table 7. We derived the extracted values by using the following relationships:

percent long reads = (number of 68000 reads − number of 68020 reads) / number of 68020 reads;
percent long writes = (number of 68000 writes − number of 68020 writes) / number of 68020 writes.

It is not possible to directly attribute the decrease in the number of instruction accesses to the wider bus. Since all instruction fetches are treated as long-word accesses on the 68020, the number of instruction accesses per instruction should be half that of the 68000, or 0.831 when the instruction cache is disabled. However, since on the 68020 the depth of the instruction pipe was increased from two words to three, and the bus controller was designed as an autonomous controller, the relationship is very complex and no useful comparisons can be made.

The percentage of total accesses that each type of access represents is shown in Table 8. While the number of data cycles per instruction is constant for the 68020, the percentage of total accesses that they represent increases as the instruction cache hit rate increases. The percentage of accesses, by type, has implications for the design of an external cache. For example, if a write-through policy is adopted, any write is treated as a miss in the off-chip cache. Thus, as the hit rate of the internal cache increases, the incremental performance benefit provided by an external cache with a write-through policy decreases, since writes comprise a greater percentage of all accesses and are treated as misses in the external cache. Knowing these data and the system behavior characteristics enable one to make the appropriate design decisions.

Performance improvements within the processor. The performance of the 68020 as shown in Table 1 is about four times the performance of the 68000 when both processors are measured at their base frequency. The primary factors contributing to this performance improvement can be divided into two categories: higher clock frequency and reduced dependence on the bus. Each category's contribution to performance complements that of the other. If memory can be accessed very quickly with minimal wait states, the clock becomes the predominant contributor to performance. If memory is slow, factors that reduce bus utilization—such as an instruction cache and a wider bus—become more important.

Memory access time is one of the major obstacles to increasing the performance of the 68020 beyond

*The only other variation in the number of data cycles is between the 68000 and the 68010, where the number of data reads is reduced for the 68010. The 68000 has superfluous fetches associated with the CLR (clear) and MOVE SR, 5EA[5] (move status register) instructions—in these cases, it fetches data before the internally generated value is stored in memory. These superfluous reads were eliminated on the 68010.

Table 8.
Breakdown of accesses by type.

Access type	68000	68010	68020 0% hit	68020 64% hit	68020 100% hit
Instr. read	62%	63%	60%	48%	0%
Data read	24%	23%	24%	32%	61%
Data write	15%	15%	15%	20%	39%
	100%	100%	100%	100%	100%

what it already provides. Although it is possible to increase the clock frequency, the performance benefit, unless accompanied by decreased system access time, will be of diminished value. Methods to decrease access time—faster buses, faster memories, and more sophisticated cache organizations (i.e., a more complex write policy)—will become increasingly important.

We can expand Equation 2 to determine the effect of memory speed on performance at various clock frequencies by considering the number of wait states:

NWS (clocks/access) = (round up (MAT (microsecs) * CR (clocks/microsec))) − 2
TCPI (clocks/instr) = BCPI (clocks/instr) + (AAPI (accesses/instr) * NWS (clocks/access))
MIPS (instrs/microsec) = CR (clocks/microsec) / TCPI (clocks/instr)

(Equation 3)

where AAPI = the average number of accesses per instruction (from Table 6), BCPI = the basic clock rate per instruction (from Table 5), CR = the clock rate (in MHz), MAT = the memory access time from address to data valid (in microseconds), NWS = the number of wait states per access, and TCPI = the total clocks per instruction.

The number of clocks per instruction, TCPI, is the sum of the basic internal execution time (BCPI is about 7), which is fixed, and the average memory delay, which is a product of the number of accesses and the number of wait states. For a given long memory access time, the primary factor determining performance is the number of accesses made. Thus, any reduction in the number of these accesses can result in significant performance improvements with slower memories.

The relationship between memory access time and performance can be seen in Figure 1, which shows the projected performance of the 68020 with its existing bus and instruction cache but with clock frequencies of 25 MHz and 32 MHz. (This example does not imply that Motorola has any plans to provide the 68020 at these frequencies. We provide the example only to indicate what the performance would be with the current bus and cache configuration. If a much higher clock frequency could be obtained, it would probably be accompanied by an increase in the size of the instruction cache and by a faster bus.) From Figure 1, it is apparent that performance is not a function of clock rate alone. If steps are not taken to reduce system access time as well, performance increases will be limited.

We should note that because the time required to access external resources is so important to performance, we selected system access time as the dimension complementary to performance for the plotted information. This enabled us to study performance through a range of system access times. System ac-

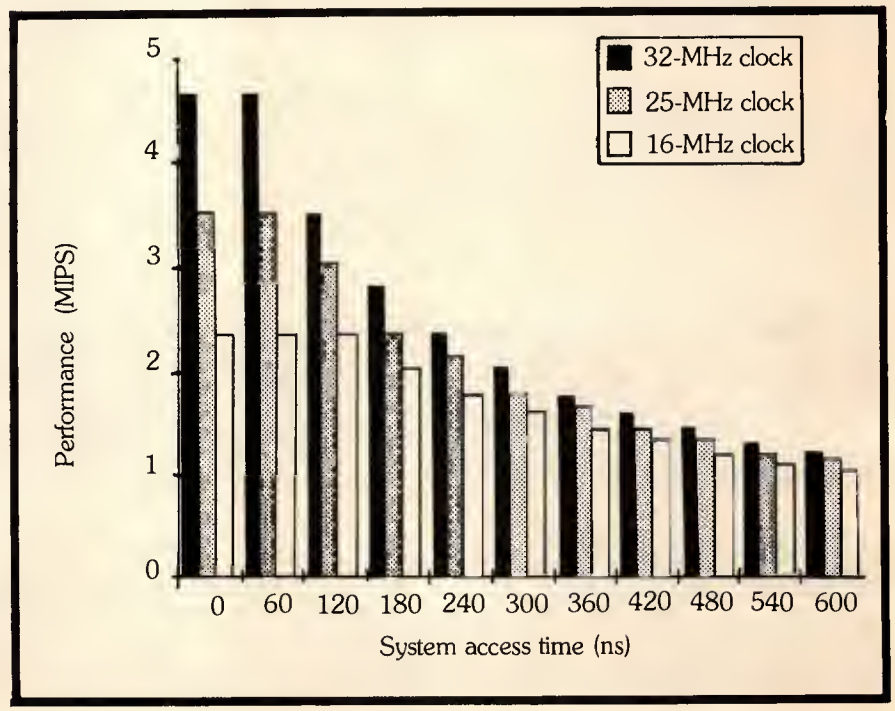


Figure 1. Performance of the 68020 at three clock frequencies—16, 25, and 32 MHz.

cess time is the time from address valid to data valid. As such, it may be dependent on system characteristics such as memory organization, address translation time, buffer and bus delays, memory access time, and error correction logic delays.

By allowing three-clock bus cycles, systems capable of performing fast memory accesses can realize a 15-percent performance improvement. Normally they achieve such a fast system access time with an off-chip cache. Figure 2 shows the performance benefit associated with three-clock bus cycles for a 16-MHz 68020 with an internal cache hit rate of 64 percent.

The 256-byte instruction cache reduces dependence on the bus by storing portions of the instruction

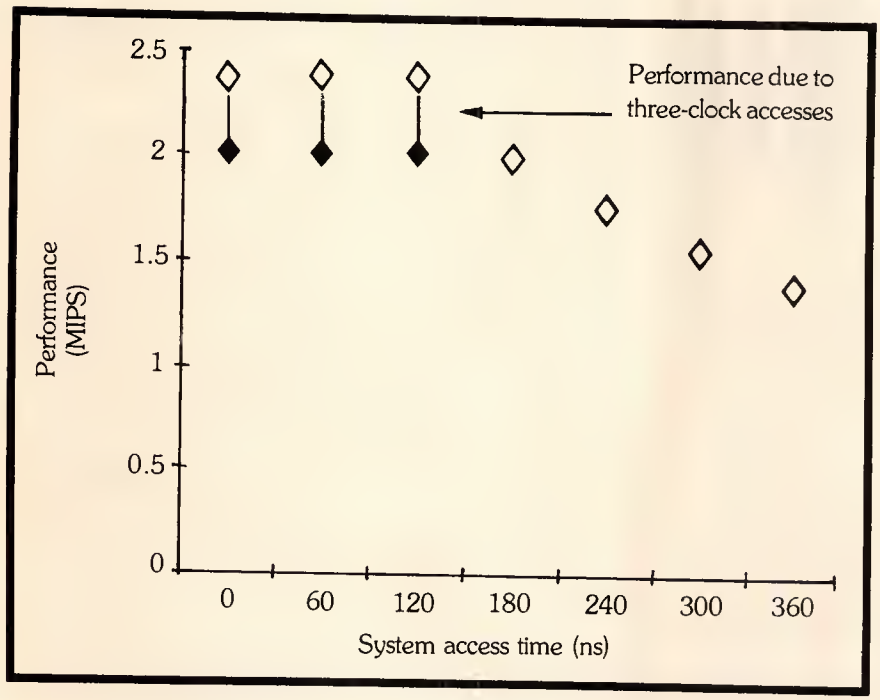


Figure 2. Performance arising from use of three-clock bus accesses.

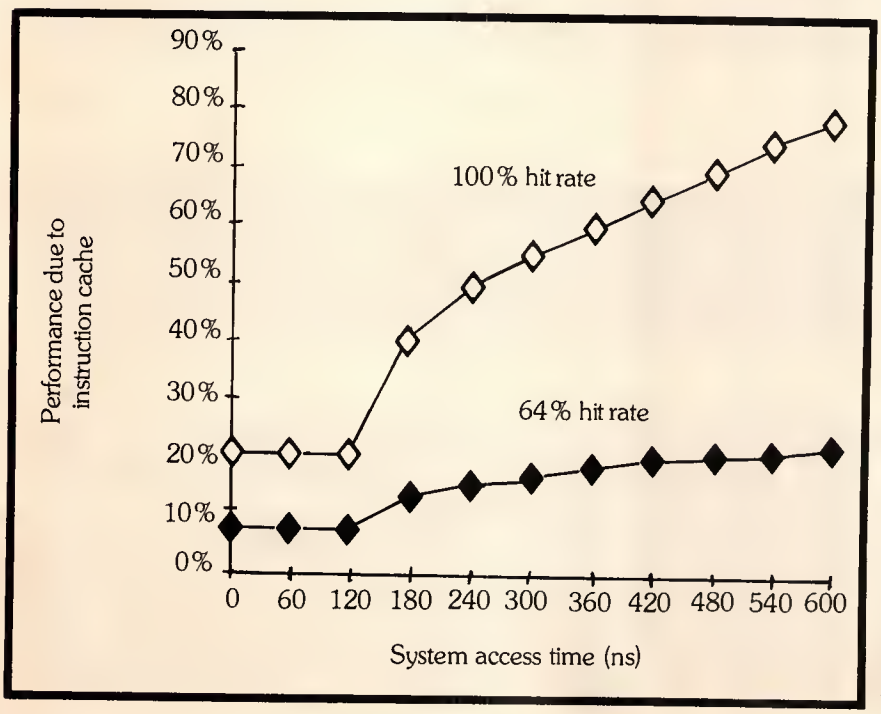


Figure 3. Performance arising from use of an instruction cache.

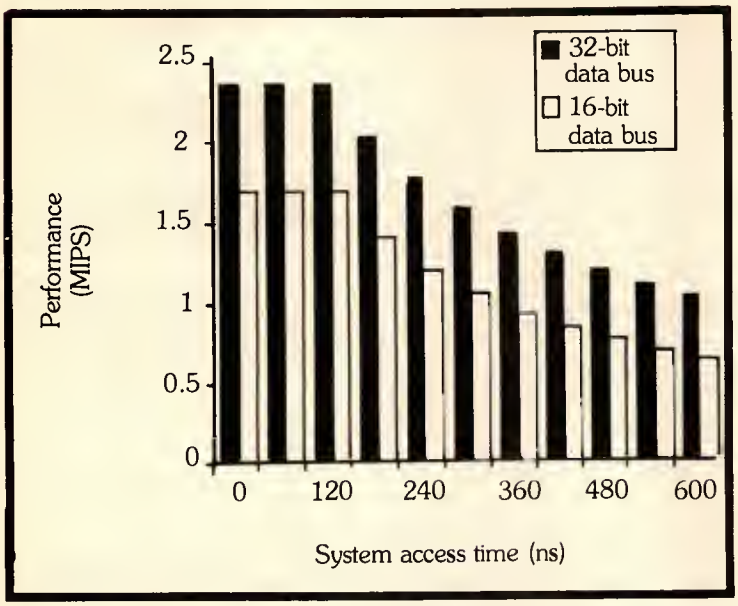


Figure 4. Performance arising from use of a 32-bit data bus vs. that arising from use of a 16-bit data bus.

stream, fetched on a demand basis, internally. By using pipelined control of the instruction address generation unit and a separate data and adddress path to the cache, the 68020 is able to access the cache completely in parallel with external bus activity. This results in effective access times of zero clocks for cache hits. The simulated cache contribution to performance is shown in Figure 3. It can be seen that as system access time increases, the cache plays an increasingly important role in 68020 performance. This trend complements the improvement in performance due to clock frequency, which decays as system access time increases. The cache provides a very fast internal memory that is independent of external system configurations and an independent accessing mechanism for the largest class of accesses, instruction fetches. By reducing the processor's dependence on the bus, one ensures that critical data accesses are not delayed and that other factors like instruction overlap can make a greater contribution to performance.

The 32-bit data bus allows long-word data accesses and instruction fetches to be performed in a single bus access. This results in a reduction in bus activity and higher performance (Figure 4).

The behavior of the 68020 in a 16-bit system is of particular interest because the 68020 has a dynamically sized data bus that enables it to run on an 8-, 16-, or 32-bit bus or on a combination of any of the three. This feature allows the system designer to implement a 68020 processor card in a 16-bit 68000 or 68010 system and realize an immediate performance benefit. If the application demands it, a 32-bit external cache can be added to bring the performance close to the 32-bit optimum. Table 9 shows the bus activity characteristics of the 68020 on a 16-bit bus.

The performance benefit arising from other enhancements is much more difficult to analyze. Some enhancements, like the new instructions and addressing modes, require an architectural analysis that is beyond the scope of this study. Others, like instruction overlap, require instruction sequence data that are difficult to collect for large traces.

Correlation of estimated to measured performance. To correlate the estimated performance to

Table 9.
16-bit bus access characteristics.

	68000	68010	68020 0% hit	68020 64% hit	68020 100% hit
Clocks/instr.	12.576	12.107	11.676	9.975	7.424
Instr. reads/instr.	1.662	1.662	2.036	1.222	0.000
Data reads/instr.	0.644	0.602	0.590	0.590	0.590
Data writes/instr.	0.392	0.392	0.383	0.383	0.383
TOTAL ACCESSES/INSTR.	2.698	2.656	3.009	2.195	0.973
% instr. reads	62%	63%	68%	56%	0%
% data reads	24%	23%	20%	27%	61%
% data writes	15%	15%	13%	17%	39%

the real performance of the 68020 chip, one must first measure the latter. For the reasons we discussed in our introductory remarks, dealing with performance in relative terms allows an unobstructed view of chip performance without interference from system-related factors. To provide the same unobstructed view during absolute measurements, we must strip from the test programs all operating system calls and traps associated with I/O, virtual demand paging, and file operations. Under these restrictions, however, we can use only very small and simple programs to determine chip timing. It is ironic that because of the simplicity of these programs the physically measured performance of the chip may be less accurate (for typical processing) than the estimated performance, which can be based on much more sophisticated and representative programs.

To measure the performance of the chip, we set up a test environment in which the clock, bus size, and number of wait states could be very accurately controlled and measured. Our test system had a limited amount of memory—4K bytes. All system functions were stripped from the program, and assembly timer routines were added. The programs were compiled into intermediate code, which was then assembled, loaded, and executed. Because of the limitations mentioned above, only three simple programs from the 20 programs analyzed could be measured. (Two of the programs were compiled in different languages. Thus, their compiled code was significantly different.)

To use these very limited data to correlate real to estimated performance, we needed to understand the nature of these programs and then make the comparisons very carefully. To do this, we determined the behavior of the programs by simulation and analysis. The difference between the nature of the programs and the composite model of bus activity we developed earlier in this article is shown in Table 10. The first column contains data describing the nature of the programs measured, while the second column shows data for the composite model. In each category of access the measured programs use the bus more than would be normally expected. In most cases the difference is small. The notable exception is the frequency of data writes—they occur much more frequently in the programs than would be normally expected. As the cache hit rate rises and writes become a greater percentage of all accesses, the correspondence of the measured programs to the model decreases. The additional bus cycles suggest that the measured performance will be less than that which will be encountered during execution of normal programs on the chip.

Table 10.
Accesses per instruction.

	Measured	Composite	Difference
Data reads	0.411	0.384	7%
Data writes	0.341	0.242	41%
0% hit instr. reads	1.035	0.958	8%
64% hit instr. reads	0.621	0.575	8%
0% hit total	1.787	1.584	13%
64% hit total	1.373	1.201	14%
100% hit total	0.752	0.626	20%

Table 11.
Estimated vs. measured performance in MIPS.

	Estimated	Measured	Est./meas.
Cache enabled (100%)	2.63	2.97	12.8%
Cache disabled (0%)	2.16	2.27	4.9%

Because these programs are very short and highly iterative, the resulting cache hit rate approaches 100 percent. This difference can be easily accommodated during the performance comparisons by replacing the normally expected hit rate of 64 percent with the 100-percent hit rate figures from the simulator.

Once we understood the relationship between the model of computation and the test programs, we were able to make a comparison between the estimated and measured performance for those programs.

Table 11 shows the correlation of the estimated to the real performance for the processor with both the cache enabled and the cache disabled. The results show that when the cache is disabled, the estimated performance is about five percent under the actual. This is natural, since several factors benefitting performance could not be considered by the method of performance estimation we used, again for the reasons we discussed in our introductory remarks.

The cache-enabled performance is significantly higher than the estimated performance (almost 13 percent higher). While we could not determine the reason for this difference with certainty, we felt that the performance improvement provided by the cache

further increases the benefit of instruction overlap, which is not considered by the simulator. As the cache hit rate increases, the bus is utilized much less, resulting in reduced conflicts and allowing more than the normally expected amount of overlap to occur.

The estimated vs. measured performance benefit resulting from the instruction cache is shown in Table 12. This comparison reflects the only major discrepancy between estimated and observed performance that we found. The measured benefit of the cache was much higher than the estimated one. Instead of the 22-percent benefit expected, the observed benefit was over 30 percent. We were not able to determine exactly why this was so, but we think that the reduced bus utilization associated with the very high hit rates must interact with other mechanisms—like instruction overlap—to produce significantly higher performance than expected.

We also measured the performance of the processor on a 16-bit bus. A comparison of the performance on a 16-bit bus to that on a 32-bit bus indicates the benefit arising from 32-bit operations for the test programs. Table 13 shows the results of this comparison for both the cache-enabled and the cache-disabled case. The performance relationship between 16- and 32-bit bus operations for the cache-disabled case is very close to the relationship we would expect to see, given the data in Table 4 and Table 9. Both estimated and measured values are shown in Table 13. Again, the estimated behavior is very close to the measured behavior, and the difference can be accounted for by an underestimation of performance.

Table 12.
Benefit resulting from cache.

	Cache disabled	Cache enabled	Benefit
32-bit estimated perf.	2.16	2.63	22%
32-bit measured perf.	2.27	2.97	31%
16-bit measured perf.	1.43	2.47	72%

Table 13.
Benefit resulting from a 32-bit bus.

	32-bit	16-bit	Measured 32-bit benefit	Estimated 32-bit benefit
Cache disabled (0%)	2.27	1.43	58%	51%
Cache enabled (100%)	2.97	2.47	20%	16%

The comparisons above show a very close correlation between the estimated and actual performance of the 68020 chip. In addition to the correlation, the test programs provide an excellent example of the danger in assuming the validity of what can be observed. In our case, only simple programs are suitable for execution in a test system in which the processor can be observed independently. The hit rate of the instruction cache, and thus the performance of the chip, is much higher for the programs observed than would be normally expected. The observed performance of the chip is 2.97 MIPS. This figure, while useful for verifying the validity of the simulation method, is not a good general performance figure. The best performance figure can be obtained by multiplying the estimated performance value (from Table 5) by an adjustment factor. This adjustment is obtained by scaling the error rate shown in Table 11 (5 to 13 percent) by the expected hit rate (64 percent). Thus, the expected performance of the 68020 is given by

$$2.33 \text{ MIPS} * 1.08 = 2.52 \text{ MIPS}.$$

While this figure still does not reflect the enhancements made to the architecture, it serves as a good estimate. We will measure the performance improvements attainable from architectural enhancements once enough different 68020 compilers are available to determine the new nature of processing.

External performance factors

Although possible, it is unlikely that a 68020 application would require no interaction external to the device (i.e., a 100-percent instruction cache hit rate and no data accesses), since if it did no useful work could be accomplished. Unfortunately, every time the processor executes an access, it is subject to a degradation in performance. The magnitude of this degradation is application-dependent and is based on the hit rates of the instruction cache and the external cache, if one is used, as well as on the memory access time.

In the previous sections, we assumed that the processor had a no-wait-state memory. Again, though

possible, it is unlikely that a cost-competitive general-purpose system would have no wait states and yet would operate at 16 or 20 MHz. Here, we investigate the performance of the 68020 processor when it is utilized in a general-purpose computing environment.

Cache simulation description. To determine system performance, one must understand the system's behavior while it is running programs that are typical of the target application. The best way to characterize a system without actually building it is to simulate it. To obtain the most accurate cache simulation results, one should use actual data from an existing system. Since the 68020 is, from an architectural point of view, an enhanced 68000, one can simulate its behavior by using real data from a 68000.

We obtained the 68000 data traces by monitoring the system bus of an HP Model 236 computer. We used a special bus interface card to monitor transactions across the bus and allow a second Model 236 computer to store the data. By buffering the bus data on the interface card, we made it possible for the monitored system to operate at half speed.

The data stored by the monitoring system consists of the 23-bit address, function codes, the upper and lower data strobes, the R/W signal, and the bus grant acknowledge (BGACK). A total of 16.7 million accesses can be stored per sample. This number of accesses equates to about 6 to 10 seconds of actual system execution time. By storing such a large number of transactions, we could characterize a multitasking system without having to average several smaller traces. In addition, our results were more accurate, since we did not need to estimate operating system overhead, as had to be done in other studies.

Once a 68000 address trace is collected, it must be converted into a 68020 trace so that the external cache hit rate can be simulated. We wrote a program to make this conversion and store a 68020 trace. This program simulates the instruction pipe of the 68020; an instruction fetch is converted into a 32-bit aligned address and is placed in the pipe. Once the instruction is fetched, the address is also placed in the simulated instruction cache. When a context switch occurs in a multitasking operating system, the instruction cache is purged. One error with this algorithm is that 68000 prefetches are not thrown away. Although we do not discuss this error here, we found that its magnitude was less than one percent. Thus, it can be ignored.

To convert data fetches from a 16-bit 68000 into those of a 32-bit 68020, all data accesses to consecutive words must be combined; the conversion program assumes that all data are aligned on 32-bit boundaries. This assumption is justified by something a 68020 compiler does for performance reasons—it forces such an alignment in most cases. Note that the combination of all consecutive 16-bit accesses introduces a small amount of error into the simulations, since two separate 16-bit accesses can be erroneously combined.

Once a 68020 address trace is available, one can simulate the behavior of an external cache. We wrote a cache simulation program to allow various parameters such as cache size, line size, set size, replacement algorithm, and associativity to be varied.

Since the hit rate of the cache is dependent not only on the cache configuration but also on the specific application, it is necessary to have different traces for different applications. In order to characterize the HP Series 200 family, we traced the three operating systems available for that series—HP stand-alone interpreted Basic, HP Pascal Workstation, and HP-UX (UNIX).

A total of 12 traces were generated—one from Basic, two from Pascal Workstation, and nine from HP-UX. The trace from Basic was a fast Fourier transform with graphics display. Since Basic is interpreted and stand-alone, the cache hit rate is less dependent on the application program than it would be with other operating systems. The interpreter uses many of the system resources and thus a larger sample was not required. We chose two types of tasks from Pascal Workstation: a large compile, which is representative of general processing, and a recalculation of a Visicalc spreadsheet, which is compute-bound.

To characterize HP-UX, we selected a total of thirty programs. We used several applications to generate the HP-UX traces:

- floating-point and non-floating-point-intensive arithmetic programs such as BlD and Sieve,
- the Pascal, Fortran, and C compilers,
- the VI editor,
- graphics-intensive programs,
- disk-intensive programs such as FIND and GREP, and
- other system utilities such as NROFF.

We ran the HP-UX programs in both single-tasking and multitasking environments and also in a variety of combinations, in an effort to characterize both I/O-bound and compute-bound systems. With all twelve traces, we analyzed a total of 180 million accesses.

Instruction cache simulation results. The hit rate for the 68020 instruction cache can be expressed in two ways: as the number of hits divided by the number of accesses, or as the number of hits divided by the number of instruction accesses. The 68020 fetches instructions 32 bits at a time, but the internal instruction width is 16 bits. Two internal instruction fetches are completed with each external instruction access. Thus, each instruction access is actually 16 bits, and data fetches are either 8, 16, or 32 bits. The hit rate is defined as

(number of hits / number of accesses) * 100.

Table 14 shows the hit rate of the instruction cache under various conditions.

Table 14.
Instruction cache hit rates.

Description	All accesses	Instruction accesses
Average of all traces	42.0	64.2
(std. dev.)	6.3	10.6
Basic system	39.4	50.5
Pascal WS	41.7	67.5
HP-UX average	42.2	65.0
(HP-UX std. dev.)	7.0	10.8
Worst hit rate	33.8	50.5
Best hit rate	58.9	89.8

As can be seen from the table, the hit rate of the 256-byte instruction cache has a significant effect on the performance of a system. Bus utilization is reduced because only 58 percent of all fetches require off-chip accesses. In addition, 64 percent of instruction fetches can be supplied in two cycles by the instruction cache and, because of the instruction pipelining, these two cycles can be completely overlapped with instruction execution. It should be noted that the instruction cache changes the normal ratios of data accesses to instruction fetches, and of reads to writes. Changes in these ratios change the expected behavior of an external cache and reduce the locality of references.

External cache simulation results. An investigation simulating all possible cache organizations would have been overwhelming. Instead, we chose a few of the most likely cache organizations. In the following discussion, a "write through" main memory update policy is assumed; therefore, all writes are considered misses. Although "copy back" provides a higher hit rate, most lower-cost systems use write through. In addition, cache loading only on demand is assumed, as is no fetching on write.

The state of the art in commercial high-speed static RAM dictates the most likely cache organizations. The most common high-speed static RAM today is a 16K-bit part organized as 2K × 8 or 4K × 4. If a processor data path of 32 bits and a set-associative organization are assumed, the smallest cache would be 8K bytes with a set size of one (direct mapped). With additional RAM, a 16K-byte cache with a set size of one or two would be convenient. (The set size would depend on which organization of RAM were used.) The largest cache size likely to be used with

Table 15.
External cache hit rates.

Cache size Set size	8K bytes SS = 1	16K bytes SS = 1	16K bytes SS = 2	32K bytes SS = 1
Average of all traces	65.3	70.8	73.6	74.9
(std. dev.)	6.2	6.1	6.3	4.9
Basic system	76.7	84.0	86.7	85.0
Pascal WS	65.9	71.7	74.2	76.1
HP-UX average	63.9	69.2	72.0	73.5
(HP-UX std. dev.)	5.4	4.7	5.3	4.1
Worst hit rate	58.0	65.1	67.0	69.2
Best hit rate	76.7	84.0	86.7	85.0

the 68020 is 32K bytes with a set size of one. We selected these four cache organizations to characterize the performance of an external cache.

Table 15 shows the cache hit rates for the four organizations with a line size of four bytes. The hit rate was calculated as the number of hits divided by the total number of accesses coming from the 68020, multiplied by 100. For the larger caches, the hit rate becomes more dependent on the percent of writes since these accesses are always counted as misses. Note that the hit rates are lower than those documented for many systems in the past because the instruction cache contains a significant portion of the instruction stream and thus reduces the locality of external accesses.

Increasing the line size of a cache usually increases its hit rate. Table 16 shows the average hit rates of caches with various line sizes. A line size of more than 32 bits implies that the data path from cache to memory is also larger than 32 bits, or that the system is implemented with a block-transfer bus such as the VMEbus[12] or with one of the newer standard microprocessor buses. It can be seen that the larger line sizes increase the hit rate. Unfortunately, when a block-transfer bus is used, the miss penalty increases with the larger line sizes. This can be avoided by increasing the width of the data path to memory; however, this form of implementation tends to be costly and is not likely to be used in a system with a 68020 main processor.

Average access time calculation. To calculate a MIPS value using the equations we defined earlier, we must first calculate the average access time of the processor. This access time is defined as the average time from when the address is valid to when the data is sampled. This time is based on the minimum access time of the processor degraded by the access time to memory or cache. Given enough data, we can calculate the access time very precisely; however, in the following discussion we have made many simplifying assumptions.

Given a full 32-bit 68020 implementation with a cache and a 32-bit data path to memory, and assuming that the access time to cache or memory is constant for a read or write, we can calculate the average access time (AAT) of the processor as follows:

$$AAT = Ehr * Cacc + (1 - Ehr) * Macc$$

(Equation 4)

where Ehr = the external cache hit rate/100, Cacc = the cache access time, and Macc = the memory access time.

If a system is implemented with a block-transfer bus such as the VMEbus or with a data bus wider than 32 bits (thus increasing the line size), the cache hit rate will increase (see again Table 16). However, the penalty for a miss will also increase. In this case, the average access time is given by

$$AAT = Ehr * Cacc + (1 - Ehr) * (\%Wr * Wacc + (1 - \%Wr) * Lacc)$$

(Equation 5)

where %Wr = the percent of misses that are write accesses / 100, Wacc = the memory write access time, and Lacc = the line access time. Note that for a line-size data bus Lacc = Racc, the memory read access time, and that for a 32-bit block transfer Lacc = Racc + Sacc * ((Ls/4) − 1), where Sacc is the subsequent read access time and Ls is the line size in bytes.

Table 16.
External cache hit rates with line size greater than four bytes.

Cache size Set size	8K bytes SS = 1	16K bytes SS = 1	16K bytes SS = 2	32K bytes SS = 1
Line size 8 bytes	71.8	75.0	76.7	77.4
(std. dev.)	5.4	5.4	5.4	4.6
Line size 16 bytes	75.1	77.2	78.5	78.9
(std. dev.)	4.9	4.9	4.9	4.4
Line size 32 bytes	76.8	78.4	79.5	79.7
(std. dev.)	4.6	4.7	4.6	4.3

Since many systems are already implemented with 68000-family processors, it is likely that the 68020 will be used in a system with a 16-bit data bus. While a 16-bit implementation will not be reduced to half the performance of a full 32-bit one, it will nevertheless suffer significant degradation. However, by providing an external 32-bit cache one can minimize this effect and achieve a significant performance increase over the rest of the 68000 family. The access time of a 68020 in a 16-bit implementation with an external 32-bit cache is given by

$$AAT = Ehr * Cacc + (1 - Ehr) * (\%Wr * (\%Wwr * Wacc + (1 - \%Wwr) * (2 * Wacc + Bded)) + (1 - \%Wr) * (2 * Racc + Bded))$$

(Equation 6)

where %Wwr = the percent of writes less than 32 bits / 100, and Bded = the bus dead time between cycles.

Of course it is possible to place the 68020 directly on a 32-bit, 16-bit, or 8-bit bus. In all three cases the average access time is the normal access time to memory, and in the 16-bit and 8-bit cases the 68020 executes additional accesses to complete the entire 32-bit fetch. For a 16-bit system with no cache, the average access time (AAT) is given by

$$AAT = (\%A132 * (2 * Macc + Bded)) + ((1 - \%A132) * Macc)$$

(Equation 7)

where %A132 = the percent of accesses less than 32 bits/100, and Bded = one clock cycle, minimum. We should note that using Table 9 to calculate a MIPS value for this configuration is easier.

Performance estimates and results

Using the above equations to determine average access times and using the MIPS value equations, one can calculate the relative performance of various 68020 implementations. Once one has an estimated MIPS value for a 68020 implementation, one can compare it to a known MIPS value for an existing 68000 system. And after this performance relationship has been established, one can use it to compare the 68020 system with other systems, since the 68000 in various implementations has been benchmarked against many systems.

Calculation of MIPS value. To actually calculate the average access time, one must make some assumptions about a typical system. First, assume a line size of four bytes unless noted otherwise and a memory access time of 540 nanoseconds. Assume also that the time between bus cycles is 120 nanoseconds, and that the access time to cache is 120 nanoseconds (three access cycles at 16.67 MHz). In the cases in which the line size is greater than four bytes, assume a 10-MHz synchronous bus. Thus, the access time can be 500 nanoseconds for the first four bytes and 100 nanoseconds for each additional four bytes. From the 68020 cache simulations, we found that with a 64-percent instruction cache hit rate, 17.4 percent of all 68020 accesses are writes, of which 37 percent are of byte or word size. We should note that these figures closely agree with the results from the instruction simulations (see Tables 7 and 8). Using the above data and Equations 4 through 7, we obtain the average access time for the 68020 (Table 17).

Table 17.
32-bit (unless noted otherwise) average access times in nanoseconds.

Description	No cache	8K bytes SS = 1	16K bytes SS = 1	16K bytes SS = 2	32K bytes SS = 1
32-bit data bus	540	266	243	231	225
16-bit data bus	540(16)	452	392	362	348
32-bit block transfer bus:					
line size 8 bytes	---	238	223	214	211
line size 16 bytes	---	237	223	214	211
line size 32 bytes	---	249	231	219	217

In addition to average access time, bus utilization has a significant effect on system performance. If a processor uses 100 percent of the available bus bandwidth, other devices on the bus (such as a DMA controller) will be locked out or the processor accesses will be held off, thereby decreasing system throughput.

From Table 17, we can see that the larger the cache, the lower the average access time. We can also see that using a block-transfer bus decreases the average access time. Using a block-transfer bus also decreases bus utilization, since there is less overhead, on the average, for each transfer. Given the assumptions upon which Table 17 is based, we can state that the lowest average access time is provided by a block-transfer bus with a line size of 8 or 16 bytes, and the lowest bus utilization by the 16-byte line size.

These calculations ignore techniques such as overlapping write accesses to memory with accesses to the cache or, in a block-transfer system, allowing the processor to continue before the entire line is fetched. Using these and other techniques, one can further reduce the access time.

Using the average access times from Table 17 and an instruction cache hit rate of 64 percent (see Table 14), we can calculate the MIPS value from the data in Tables 4, 6, and 9:

MIPS = (clock freq / (avg clocks/instr + (avg bus cycles/instr * ((AAT/clock period) − 2)))) * 1.08

(Equation 8)

Table 18 shows the MIPS values calculated from Equation 8 for the 68020 system configurations we have been discussing.

As can be seen, the performance of the 68020 spans a large range. The lowest-performance system presented is one using a 16-bit bus with no external cache. Since an 8-MHz 68000 with an equivalent access time has a performance of about 0.45 MIPS, the 68020 can provide a 70-percent increase in performance with little additional cost to the system. By adding a large external cache, one can obtain a significant performance increase.

Actual system results. The HP Series 200 is based on the 68000 and 68010. To extend this series, HP has developed a 68020-based system as part of its Series 300 line. To obtain the highest performance/price ratio and give the most value to the customer, HP provided this system with a full 32-bit cache and a 16-bit data bus to memory. Its 68020 CPU board is compatible with the memory and I/O originally used in the Series 200 family.

Although the actual parameters of the new system do not exactly match the numbers we used in our average access time calculations, they are close enough that a valid comparison can be made (our average access times are slightly less). The 68020 CPU board is implemented with a 16K-byte cache, with a set size equal to one and a line size of four bytes. Processor accesses to cache are three cycles total (120 nanoseconds, address to data). Accesses to memory are either byte or word in length, and two fetches are executed to fill a line in the cache. Given these characteristics, we obtain a performance of around 1.5 MIPS for the new HP system.

Table 19 compares the measured performance of the new 68020-based system with that of an 8-MHz 68000 system with one wait state and that of an 8-MHz 68010 system with 1.5 wait states (0.5 of a

Table 18.
16.67-MHz 68020 MIPS values.

Description	No cache	8K bytes SS = 1	16K bytes SS = 1	16K bytes SS = 2	32K bytes SS = 1
32-bit data bus	1.16	1.79	1.87	1.92	1.94
16-bit data bus	0.80	1.30	1.43	1.50	1.54
Line size 8 bytes	---	1.89	1.95	1.99	2.00
Line size 16 bytes	---	1.89	1.95	1.99	2.00
Line size 32 bytes	---	1.85	1.92	1.97	1.98

Table 19.
Benchmark execution times in milliseconds.

Benchmark name	OS	8-MHz 68000/10	16-MHz 68020	16-MHz 68020 16-MHz 68881
Sieve	PWS	679	158	
Acker	PWS	5660	1730	
Puzzle	PWS	23410	5450	
Search	PWS	3.4	0.8	
B1D	PWS	376480	105000	14700
Pipes	HP-UX	9000	4200	
Scall	HP-UX	11900	4700	
Fcall	HP-UX	1300	400	
Loop	HP-UX	11500	2700	
B1D Fortran	HP-UX	413500	113500	
LFP Fortran	HP-UX	2079600	732900	

Table 20.
Benchmark relative performance.

Benchmark name	OS	8-MHz 68000/10	16-MHz 68020	16-MHz 68020 16-MHz 68881
Sieve	PWS	1	4.3	
Acker	PWS	1	3.3	
Puzzle	PWS	1	4.3	
Search	PWS	1	4.3	
B1D	PWS	1	3.6	25.6
Pipes	HP-UX	1	2.1	
Scall	HP-UX	1	2.5	
Fcall	HP-UX	1	3.3	
Loop	HP-UX	1	4.3	
B1D Fortran	HP-UX	1	3.6	28.5
LFP Fortran	HP-UX	1	2.8	6.0
Avg. performance:		1	3.49	---
HP-UX avg. perf.:		1	3.10	

wait state for the MMU). The 68000 and 68010 systems ran identical benchmarks. The 68000 benchmarks were run in Pascal on the Pascal Workstation (PWS). The 68010 benchmarks were run in C and Fortran on HP-UX. All benchmarks shown use 32-bit integers and 64-bit reals.

In choosing benchmarks to compare processor performance, we preferred compute-bound programs since I/O throughput tends to be disk-, not processor-, dependent. The integer benchmarks (Sieve, Acker, Puzzle, and Search) were collected by Hansen et al.,[13] and results have been published for several systems.[14] The UNIX benchmarks were published in *Byte*,[15] and the results are in real times. BlD is a standard Whetstone floating-point benchmark (10 million executions), and LFP is a very large compute-bound, floating-point program.

The performance of an 8-MHz 68000 system with one wait state or an 8-MHz 68010 system with 1.5 wait states is about 0.5 MIPS. The calculated MIPS value for the new HP 68020 system is 1.5, which is about three times faster than the 68000/10 systems. The benchmark comparisons show that the measured average relative performance increase is 3.49 (Table 20). Note that for the small integer benchmarks, the 68020 has an unfair advantage in that the programs can be loaded entirely into the internal and external cache. Thus, even though many simplifying and conservative assumptions have been made and many errors have been introduced into the simulation results, the measured performance exceeds the calculated results by only 10 to 15 percent.

As always, the choice of benchmarks can dictate how well a system fares when it is compared to other systems. Our intention in choosing the benchmarks presented here was not to fully characterize the performance of a system but to correlate theoretical results with the results from an actual system. We leave it to the various computer manufacturers to characterize and represent their systems.

We have attempted to provide insight into the raw potential performance of the 68020 and its performance within a system environment. A final analysis of our results shows that the estimated performance of the 68020, and of a system based on that processor, correlates closely to the measured performance. In addition, two independent simulations, one developed at Motorola and the other at Hewlett-Packard, agreed closely. Because of this correlation, we feel our results are reliable and constitute a reasonable representation of the 68020 in a general processing environment.

The measured performance of both the chip and the system was higher than the simulations predicted. While it would be easy to attribute the higher performance to some random factor and claim that the 68020 does indeed have that measured performance, that would be unfair. The difference in estimated versus measured performance can be explained by the simplicity of some of the programs used as benchmarks and by the conservative approach taken in the simulations. Unfortunately, the benchmarks commonly used to measure performance tend to lack the robustness needed to fully exercise and evaluate a system.

In establishing the performance of the 68020 relative to that of the 68000, one can expand absolute performance comparisons between the 68000 and other microprocessors[16] to include the 68020. Using the processing models developed in this article, one can compare various systems and configurations with greater accuracy and confidence.

Our study presented a unique opportunity to investigate a new processor both internally and as part of a system implementation. Our processor and system simulations were extremely useful in developing HP's Series 300 computer family—they helped ensure the highest possible performance-to-price ratio. The simulations were also extremely useful in the MC68020 development at both Hewlett-Packard and Motorola.

References

1. H. Levy and D. Clark, "On the Use of Benchmarks for Measuring System Performance," *Computer Architecture News*, Vol. 10, No. 6, Dec. 1982.
2. E. Stritter and T. Gunter, "A Microprocessor Architecture for a Changing World: The Motorola 68000," *Computer*, Vol. 12, No. 2, Feb. 1979, pp. 43-52.
3. J. Zolnowsky and N. Tredennick, "Design and Implementation of System Features for the MC68000," *Proc. Compcon Fall 79*, Sept. 1979, pp. 2-9.
4. D. MacGregor and D. S. Mothersole, "Virtual Memory and the MC68010," *IEEE Micro*, Vol. 3, No. 3, June 1983, pp. 24-39.

5. D. MacGregor, D. S. Mothersole, and B. Moyer, "The Motorola MC68020," *IEEE Micro*, Vol. 4, No. 4, Aug. 1984, pp. 101-118.
6. Motorola, Inc., *MC68020 32-bit Microprocessor User's Manual*, Prentice-Hall, Englewood Cliffs, NJ, 1984.
7. J. Retsky, *MC68020 & MC68881 Platform Board for Evaluation in a 16-bit System*, Motorola Application Note.
8. D. MacGregor, "Hardware and Software Strategies for the MC68020," *EDN*, Vol. 30, No. 14, June 20, 1985.
9. A. Smith, "Cache Memories," *Computing Surveys*, Vol. 14, No. 3, Sept. 1982.
10. D. R. Ditzel, "Program Measurements on a High-Level Language Computer," *Computer*, Vol. 13, No. 8, Aug. 1980, pp. 62-72.
11. W. Alexander and D. Wortman, "Static and Dynamic Characteristics of XPL Programs," *Computer*, Vol. 8, No. 11, Nov. 1975.
12. *VMEbus Specification Manual*, Motorola, Inc., Pub. no. MVMEBS/D1, Aug. 1982.
13. P. Hansen, M. Linton, R. Mayo, M. Murphy, and D. Patterson, "A Performance Evaluation of the Intel iAPX 432," *Computer Architecture News*, Vol. 10, No. 4, June 1982.
14. A. Gupta and H. D. Toong, "An Architectural Comparison of 32-bit Microprocessors," *IEEE Micro*, Vol. 3, No. 1, Feb. 1983, pp. 9-22.
15. D. F. Hinnant, "Benchmarking UNIX Systems," *Byte*, Vol. 9, No. 8, Aug. 1984.
16. R. Grappel and J. Hemmenway, "A Tale of Four Microprocessors: Benchmarks Quantify Performance," *EDN*, Apr. 1, 1981.

Jon Rubinstein is an R&D engineer at Hewlett-Packard's Fort Collins Systems Division. He was a member of the design team for the HP 9000 Series 300 and the Series 200 Model 236. For the Series 300 his primary responsibility was the design and definition of the 68020-based Model 320 processor board.

Rubinstein received his BS and ME in electrical engineering from Cornell University and is presently completing his MSCS at Colorado State University. He is a member of ACM and IEEE.

Questions about this article can be directed to Rubinstein at Hewlett-Packard, 3404 East Harmony Road, Fort Collins, CO 80525.

Doug MacGregor designed the control structures and wrote the microcode for the 68010 and the 68020. Aside from picking up a BA in history and Asian studies at night, he got some maturity and direction while serving in the US Navy. Deciding that eating had some attractive merits, he attended the University of Illinois, where he obtained an MS in computer science and after which he began work at Motorola. He is currently pursuing a doctorate in computer science at the University of Kyoto. His passions include acoustic analysis of submarines, etymology, and efficiency.

MicroStandards Special Feature: A Comparison of 32-Bit Buses

Paul L. Borrill
University College London
Mullard Space Science Laboratory

As promised in the October MicroStandards column, I am presenting a comparison of several well-known 32-bit buses in tabular form this month. Because of the danger of reading too much into this table, and the potential for misunderstanding, I will elaborate on many of the issues. If further clarifications are needed, contact me at Holmbury St-Mary, Dorking, Surrey RH5 6NT, England. It is intended that this subject could form an ongoing discussion in *IEEE Micro,* and user comments and experience are actively solicited to refine the understanding of the relative advantages and disadvantages of these buses, to benefit the readers of *IEEE Micro.*

While the October column concentrated on redressing the balance between the previous comments published in August by Hubert Kirrmann[1] concerning Multibus II and the Futurebus, this month I put considerable effort into producing a balanced overview of the features of more 32-bit buses. In particular, representative proponents of each of the buses were asked to comment and help refine this version. However, ultimate responsibility for any remaining errors lies with the author. The sequence of discussion in this text follows the sequence in the table.

Support. To avoid the inevitable numbers game each of the vendors tend to play when claiming multiple sources for compatible products, I present only the primary sponsor and supporters of each bus as an indication of its genealogy.

Bus bandwidth. The performance figures are calculated values taking into account realistic assumptions that are applied consistently to each bus as far as possible. Included in the calculations are bus driver and receiver propagation delays, transmission line driving delays (the bus-driving problem and signal propagation delays down the backplane), logic and backplane skews, setup and hold times for latches, and slave access times. Since a synchronous bus must have a clock period slow enough to cope with a fully loaded system, the calculations for the asynchronous buses also assume a fully loaded system, even though this may not always be true. An asynchronous bus may be faster with shorter backplanes or in a more lightly loaded typical system.[2]

The performance figures for the Nubus and Multibus II are almost identical because both use a synchronous protocol clocked at 10 MHz. The performance in the single transfer mode for Multibus II and Nubus includes some concurrency, which is possible between the address decode plus access time delays and the address/data multiplexing delays. The Nubus appears to have a faster arbitration time than Multibus II because its shorter backplane and fewer arbitration lines allow the logic to settle faster.

0272-1732/85/1200-0071$01.00 © 1985 IEEE

A simple calculation shows that the average distance between any two communicating boards on a backplane is one third of the backplane length, so this figure is used in the calculation of the backplane delays for the asynchronous protocols. Synchronous buses have a fixed clock period, which must be long enough to cope with boards communicating from each extremity of the backplane. The performance of a synchronous bus is thus independent of whatever two boards happen to be communicating at any instant in time.

Identical fully loaded backplane delays were applied to the VME bus, Futurebus, and Fastbus, even though in the case of the Futurebus this would be an overestimation because Futurebus transceivers load the bus less. Fastbus and Futurebus use essentially the same protocol, and differences in bus bandwidth and arbitration performance are due primarily to the difference in the currently available technology that implements the two; i.e., advanced Schottky TTL for the Futurebus (the same as for the VME bus), and 10-K ECL for Fastbus.

A noteworthy result is shown in the figures for single transfer mode, where all of the buses have a very similar performance. This is due primarily to the fact that the VME bus, which is nonmultiplexed, does not suffer the delays of multiplexing the address and data on the same lines as do the other buses. In addition there are transactional overheads in the protocol for the Futurebus and Fastbus besides the multiplexing: the Futurebus incurs delays required to overcome the transmission line effect required called the wire-OR glitch[3] on the fully broadcast address transfers; the Fastbus has a 40-ns clean-up delay at the end of each transaction (to solve essentially the same problem).

Address pipelining, which is possible only on the VME bus because it is nonmultiplexed, is not taken into account in the calculations because of the complexity of the assumptions needed to justify the results.

The burst transfer mode, which is the usual "communications-oriented" assumption for transactions in a typical multiprocessor system, shows a wider difference in bus performance; but at longer access times performances start to equalize rapidly.

The lesson to be learned from this is that a high-performance bus alone is not sufficient to guarantee a high-performance system. Some architectural means are necessary to utilize fully the bus's available bandwidth. For example, a method is needed to reduce either the amount of bus traffic each board generates or to reduce the "effective" bus access time. (This may be done predominantly by using blocks to communicate across the bus and by using static column address or nibble-mode access RAMs on the memory boards.) Effective ways to reduce bus traffic are (1) cache memories between each processor and the bus and (2) local memories located on each board or accessible through some kind of local extension bus. A cache can be transparent to programs whereas local memories are generally not transparent. Effective ways to force the system to pass blocks over the bus rather than single transfers are using cache memories or restricting the system to a message-passing architecture. Cache memories and message passing are discussed later.

A high-performance bus alone is not sufficient to guarantee a high-performance system.

The highly artificial assumption of processor-, memory-, and bus-clock coherence is made in the figures for the performance of the synchronous buses. The clock-latency problem would effectively add a half-cycle (50-ns) penalty to every transfer on Multibus II and Nubus, lowering their performance in many real implementations.

The clock-latency problem can be viewed as follows. Because all bus-based multiprocessor systems would be limited by the bottleneck the bus would form if the processors accessed all instructions and data directly through the bus, it is necessary to find ways to keep as much as possible of the instructions and data local to each processor so it can reduce its bus-utilization requirements. Therefore, for a large percentage of the time (90 percent, say) processors are executing and manipulating data locally, and only a small percentage of the time (10 percent, say) do they need to use the bus. It therefore becomes pragmatic to optimize the processor to its local resources rather than to the bus; i.e., choose a processor clock speed and memory access as the best economically available at the time. The 12.5- and 16.7-MHz processors are typical this year, with 20- and 25-MHz processors expected in 1986. This means that each local processor clock will be different and mutually asynchronous to the bus clock. Consequently, in the one out of 10 times the processor needs access to the bus, its interface must synchronize the two somehow to pass the data. This synchronization causes two problems: the delay needed to wait for the first valid bus clock edge, and the metastable state problems intrinsic to the synchronizer circuits.

On each transfer the processor makes through the bus, after it presents its request to the interface, the interface must wait for the first valid clock edge. The request may arrive just before a bus-clock edge is due, so there would not be time to wait, or it may arrive just after a bus-clock edge has occurred, so the processor must wait an entire cycle before another clock edge is due. On average, the delay will be half a clock cycle. This is the clock-latency problem.

Multibus II recognizes this problem by providing a method of buffering the timing of the processor from that of the bus, called message passing. Provided the implementor is prepared to accept the constraints of such an architecture, clock latency can be mostly overcome by this technique. Nubus suffers from the clock-latency problem also, but does not explicitly offer a facility, such as message passing, to help overcome it.

Asynchronous buses do not suffer the same clock latency problem, because the bus adopts the timing of the master. On the other hand, it is often claimed that synchronous buses are easier to design, are more reliable, and have less noise problems than asynchronous buses. This is a highly subjective argument, which tends to generalize too much on categories of synchronous and asynchronous buses without taking into account the details of how carefully the individual buses are designed.

Arbitration. The information needed to estimate the arbitration overhead and average bus-acquisition latency is shown in algebraic form as well as numbers in the table, since this is very application specific. T_{arb} indicates the time it takes for the arbitration system to resolve multiple requests; it is equivalent to the time it takes a master to acquire the bus when it is not in use, assuming the bus was not previously owned by that master. All the buses have an effective "parking" mode, where there is essentially zero delay to acquire the bus, if there is no change in mastership. This is called Release On Request (ROR) in VME bus terminology. Release When Done (RWD) is not normally useful ex-

Table 1.
Comparison of 32-bit buses.

This table is believed to be entirely accurate. However, it is the nature of these backblane buses that changes in their specifications, environment, and commercial support change with time. Consequently, this table will be updated periodically, and the latest version should always be obtained when the most accurate and up-to-date comparisons are sought.

General features

Aspect Bus-> V	VME bus	Futurebus	Multibus II	Nubus	Fastbus
Standardization status	IEEE P1014 Draft 1.2 IEC 821 draft	IEEE P896.1 Draft 7.2 due for release, 1Q 86	IEEE Pxxx Intel Rev. C	IEEE Pyyy Draft 1.0	ANSI/IEEE 960 Approved Standard IEC45 (Sec) 243
Primary sponsor	Motorola	IEEE P896 W.G.	Intel	Texas Instruments	US Nim Committee
Primary supporters	Signetics Mostek	Specification not yet released for commercial use		Lisp Machines Inc.	Kinetic Systems LeCroy Research Dr. B. Struck and others
Silicon support					
Now	Motorola, Signetics	National Semiconductor	Toshiba	None	Maruei Shoji Co. Several gate arrays
Expected	Hamilton Standard Digital Systems	Ferranti, Monolithic Memories, Signetics, Texas Instruments	Intel	Texas Instruments	Fujitsu, Valtronic Inc., Integrated Networks, Philips Elcoma (Zurich)
Performance	Continuously variable	Continuously variable	Quantized 100-ns wait states	Quantized 100-ns wait states	Continuously variable
Bus bandwidth (M bytes/s)					
Sponsor claims	20 to 57	117.6	40*	37.5*	160
Source	Motorola	*IEEE Micro,* Aug. 84 (boards adjacent)	Intel (assumes 10-MHz clock)	TI (16-cycle block and 10-MHz clock)	*IEEE Trans. Nuclear Science,* Feb. 85
Single transfer mode (average backplane delay = ⅓ backplane length) (M bytes/s)					
T_{acc} = 0 ns	25.0	37.0	20*	20*	37.0
T_{acc} = 50 ns	19.0	25.3	20*	20*	25.3
T_{acc} = 100 ns	15.4	19.2	13.3*	13.3*	19.2
T_{acc} = 150 ns	12.9	15.5	13.3*	13.3*	15.5
Burst transfer mode (handshaken, infinite-length block, average backplane delay = ⅓ backplane length) (M bytes/s)					
T_{acc} = 0 ns	27.9	95.2	40.0*	40.0*	173.9
T_{acc} = 50 ns	20.7	43.5	20.0*	20.0*	54.8
T_{acc} = 100 ns	16.5	28.2	20.0*	20.0*	32.5
T_{acc} = 150 ns	13.6	20.8	13.3*	13.3*	23.1
Pipelined mode (handshakes not waited for, infinite-length block) (M byte/s)	Not specified	Not specified	Not applicable	Not applicable	Limited only by skew
Ultimate future performance	~35	~280	40*	40*	~500
Message passing mode	Not defined	To be specified in P896.2	30M bytes/s	Not defined	Not defined
Fundamental limitations	Transition times, bus-driving problem, logic delays, skew, backplane prop delay, timing constraints built in spec.	Logic delays, skew, backplane prop delay	100 ns between signal sampling, due to fixed clock	100 ns between signal sampling, due to fixed clock	Logic delays, skew, segment prop delay

*Ignores clock latency

Table 1—continued

Arbitration					
Algorithms	RWD, ROR	Fair, priority	Fair, priority	Fair	Fair, priority
T_{arb} (typical ns)	200-500	250	300*	200*	150
T_{arb} (best ns)	150	100	300*	200*	90
T_m	$256.T_t$	Unconstrained	$32.T_t$	$16.T_t$	Unconstrained
T_{get} (priority)	$T_{arb}+2T_m$	$T_{arb}+T_m$	$T_{arb}+2T_m$	$T_{arb}+(N-1).T_m$	$T_{arb}+2T_m$
T_{get} (fairness)	$T_{arb}+(N-1).T_m$	$T_{arb}+(N-1).T_m$	$T_{arb}+(N-1).T_m$	$T_{arb}+(N-1).T_m$	$T_{arb}+(N-1).T_m$
Comparison with T_{arb} (typical), and T_m limited 16 transfers, in block transfer mode, 150-ns access time.					
T_{get} (best) μs	9.6	3.3	9.9	72.2	5.7
T_{get} (worst) μs	70.8	46.3	72.3	72.2	41.7

Protocol features

Aspect V / Bus->	VME bus	Futurebus	Multibus II	Nubus	Fastbus
Bus protocol	Asynchronous	Asynchronous technology-independent	Synchronous 10-MHz clock	Synchronous 10-MHz clock	Asynchronous with optional synchronous suboperations
Data path	Non-multiplexed	Multiplexed	Multiplexed	Multiplexed	Multiplexed
Primary	16 bit	32 bit	32 bit	32 bit	32 bit only
Secondary	32, 24, 16, 8 bit	32, 24, 16, 8 bit	32, 24, 16, 8 bit	32, 16, 8 bit	Not supported
Justification	16 bit justified	Nonjustified	16 bit justified	Nonjustified	Nonjustified
Nonaligned 32/16-bit-operations	Rev. C only	Fully supported	Fully supported	Not supported	Not supported
Byte orientation	Big-endian	Not constrained	Little-endian	Little-endian	Not applicable
Address spaces					
Primary (bytes)	2^{24}	2^{32}	2^{32}	2^{32}	2^{32} (quadlets)
Secondary (bytes)	2^{32}	Expandable	None	None	Expandable
Interconnect	In I/O space	CSR space**	2^{14}	CSR space**	CSR space
I/O space (bytes)	2^{16}	CSR space**	2^{16}	CSR space**	Quadlet = 4 bytes
Broadcast (writes to multiple slaves)	Not supported	Broadcast on any write transaction	Message space only, not supported in memory or I/O space	Not supported	Broadcast on any write transaction; module subset and system subset selection facilities
Broadcall (Reads from multiple slaves)	Not supported	Broadcall on any read transaction	Not supported	Not supported	Broadcall on any read Sparse data scan through T-pin
Bus repeater					
Circuit switched	Not supported	Bus repeater deadlock prevention	Bus repeater deadlock prevention	Bus repeater deadlock prevention	Fully supported
Packet switched	Not specified	Extension beyond the backplane is considered as a store-and-forward (packet-switched) operation (896.2)	Fully specified message-passing mechanism can be used for 256 nodes	Not specified	Supported, but packet-switched operations are not specified in ANSI/IEEE 960
Provision for future expansion	1 reserved line, spare address modifier states	Extended command mode, built in protocol to ensure compatibility	2 reserved lines, 1 reserved state on SC lines	No comment	5 reserved lines; protocols designed to permit expansion by additional levels of multiplexing

Table 1—continued

Locking					
Arbitration lock	Use BBSY* line and release when done	Release when done	Release when done when lock line is active	Continues request; relies on pure fairness mechanism	Release when done
Address lock	All transfers must be assumed locked	Fully supported (lock line)	Fully supported (lock line)	All transfers must be assumed locked	All transfers must be assumed locked
Single transaction lock	By AS* asserted through transaction	Fully supported (lock line)	Fully supported (lock line)	Not supported	By AS/AK lock
Multiple transaction lock	Not supported	Fully supported (unlocks on change of mastership)	Not supported	Not supported	GK lock supports multiple modules and bus segments
Bus diagnostic features					
Debugging	None	Fully handshaken broadcast mode, extender-board operation accounted for	None	None	Timeouts can be disabled; wait line can single-step transactions
Monitor/snoop	Non-handshaken address monitor timing specified	Fully asynchronous connection phase to allow inspection of address transfer; slave intervention capability	Clocked for easy logic analyzer inspection	Clocked for easy logic analyzer inspection	Wait line to slow handshake for inspection
Bus utilities available to the user	Spare address modifier codes, 64 pins free on P2 connector	All user-defined facilities are delegated to auxiliary connectors	Extra power buses +12V, −12V, +5V battery	Extra power buses +12V, −12V, −5.2V	Power buses: Standard +5, −5.2, −2.0; Optional +15, −15, +28. Daisy chain to right; daisy chain to left; eight terminated lines; two unterminated lines; four unbussed pins

**CSR space for these buses is incorporated within the main memory space.

"Not supported" means that no direct provision has been made in the bus specification for this facility. This does not mean that the feature cannot be implemented by some nonstandard means, or by future modification of the specification. However, since many of these features have significant architectural and protocol implications to the specification, the likelihood of being able to graft such features onto a specification adequately as an afterthought is remote.

Measures of cost and complexity

Aspect V / Bus->	VME bus	Futurebus	Multibus II	Nubus	Fastbus
Address spaces	64	1	4	1	2
Connectors	2	1	1	1	1
Pins	128	96	96	96	130
Number of active signal lines	107	67	67	46	60
Interrupt lines	7	0	0	0	1
Power rails	+5V, +5V SBY +12V, −12V	+5V only	+5V, +5V battery +12V, −12V	+5V, −5.2V +12V, −12V	Optional +5, −5.2, −2.0 Standard +15, −15, +28

Table 1—continued

Electrical/reliability issues

Aspect V / Bus->	VME bus	Futurebus	Multibus II	Nubus	Fastbus
Bus interface	TTL mixture 48 and 64 mA Tristate and open collector	BTL (50-mA backplane transceiver logic) (solves the bus-driving problem)	TTL mixture 48 and 64 mA Tristate and open collector	TTL mixture 48 and 64 mA Tristate and open collector	ECL 10K (darn near solves the bus-driving problem)
Parity	None	Optional	Mandatory	Optional	Optional
Inhibit control	None	Via CSR space	None	Enable line	Enable line
Parallel bus	None	1 bit per byte	1 bit per byte	1 bit for AD lines	1 bit for all
Control lines	None	1 bit	2 bits	None	None
Tag	None	1 bit	Not applicable	Not applicable	Not applicable
Arbitration	None	1 bit	None	None	None

Maintenance/configurability issues

Aspect V / Bus->	VME bus	Futurebus	Multibus II	Nubus	Fastbus
Live insertion	Not available	Fully supported with umbilical	Not available	Not available	Partially supported (halt line)
Extender card	Not allowed for	Fully supported	Not allowed for	Not allowed for	Partially supported
Geographical addressing	None	5-bit slot ID	LACHn pin (T-pin technique with power-up initialization	4-bit slot ID	5-bit 1D
Autoconfiguration	Not supported	Fully supported at any time, including live insertion	Fully supported at power-up time only	Fully supported	Fully supported

Multiprocessor support

Aspect V / Bus->	VME bus	Futurebus	Multibus II	Nubus	Fastbus
Virtual interrupts (to specific processors)	Not specified	Supported, further specified in P896.2	Supported	Supported, further specified in separate document	Supported
Arbitration	4-level daisy chain	Fully distributed, asynchronous	Distributed, synchronous central clock	Distributed synchronous central clock	Distributed central timing element
Deterministic acquisition	Supported, 4-deep round-robin system	Fully supported (fairness)	Fully supported (fairness)	Fully supported (fairness) (16 levels)	Fully supported (assured access mode—fairness)

Table 1—continued

Priority acquisition	Supported, 4 levels only, except in round-robin	Fully supported, 32 levels	Fully supported, 32 levels possible	Not supported	Fully supported (63 levels)
Message passing	Not supported	Format defined; fully specified in P896.2	Fully supported	Not supported	Not directly supported
Cache					
Write-through	Limited capability	Fully supported	Limited capability	Limited capability	Limited capability
Write-back	Not supported	Fully supported	Not supported	Not supported	Not supported
Tagged architectures (memory bits identify meaning of data)	Not directly supported; possible implementation with address modifiers	Fully supports tag bit with parity protection	Not directly supported	Not directly supported	Not directly supported

Physical features

Aspect Bus-> V	VME bus	Futurebus	Multibus II	Nubus	Fastbus
Board sizes in mm (in.)					
Primary	233.35×160 (9.187×6.3)	366.7×280 (14.437×11.024)	233.35×220 (9.187×8.661)	366.7×280 (14.437×11.024)	366.7×400 (14.437×15.748)
Secondary	100×160 (3.937×6.3)	233.35×280 (9.187×11.024)	100×220 (3.937×8.661)	None	None
Board area in cm^2 ($in.^2$)					
Primary	373 (58)	1027 (159)	513 (80)	1027 (159)	1467 (227)
Secondary	160 (25)	653 (101)	220 (34)	None	None
Connectors					
Dedicated	2 IEC 603-2	1 IEC 603-2	1 IEC 603-2	1 IEC 603-2	Multisourced AMP 2-532956, or DuPont 66527-565 or SAE RTP 2525-1303
Number of pins	2×96	96	96	96	130
Undedicated	0 (64 pins spare) IEC 603-2	2 IEC 603-2	1 IEC 603-2	2 IEC 603-2	Multisourced e.g., AMP 2-532981-1
Number of pins	64 spare on second connector	2×96	96	96	195 maximum
Number of modules					
Logical modules (maximum)	Not logically constrained	32 per backplane 65,536 message nodes	32 per backplane 256 message nodes	16 per backplane	26 per backplane segment 16,777,216×255 in fully connected system
Physical slots					
Dedicated	1	0	1	0	0
Nondedicated	20	21 (19-in. rack)	19	16	26 (19-in. rack)

cept in master-slave multiprocessor architectures, an architecture particularly well suited to the VME bus.

T_m indicates the time for mastership; i.e., the time any one master is allowed to keep the bus under normal circumstances. This is not always specified clearly for some of the buses, and others deliberately do not specify it, claiming it is the privilege and responsibility of the system implementor to set this number. For the Nubus, this limit is built into the protocol by the way it does the block transfer. For the VME bus the limit is chosen arbitrarily to ease implementation difficulties and to solve the memory-boundary problem for block transfers. For Multibus II this is also an arbitrary limit related to the maximum useful size of a message in the message-passing mode. For the Futurebus and Fastbus, the system implementor is encouraged to program limits appropriate to his system. Typically, 16 or 32 quadlets (64 or 128 bytes) will be chosen as a maximum, and the boards will be programmed to split larger blocks into several smaller ones of this maximum size. However, it may make sense in some systems to set the maximum size at 1K bytes (256 transfers) or larger to pass an entire page of virtual memory from the disk controller to the main memory, for example.

The time for a master to acquire the bus in various T_{get} intervals is also shown in the table. This time interval calculation assumes 16-word-limited blocks, 150-ns slave-access times for the transactions, and 16 modules per system. The priority case T_{get} (best) assumes there is only one high-priority master in the system. The fairness case T_{get} (worst) assumes there are 16 modules operating in fairness mode.

T_{get}(best) is the maximum time after the instant a high-priority module requests the bus in a heavily loaded system until it receives bus control. Note that T_{get}(best) and T_{get}(worst) relate to the priority of the requesting module. T_{get}(best) may be of most importance in a system processing real-time data, or, in a system with a need to react to very fast events. This needs further explanation. Clearly, if the bus is in use, even if an arbitration has taken place, the winner cannot assume control until the previous master has finished. However, there is one more complication: a simple queueing model shows that on average there will be one board that is currently using the bus, and another that has arbitrated, won, and is waiting to use the bus (the master-elect). Thus, a new high-priority master requesting the bus must not only wait for the current master to finish, but for the master-elect to finish also, even though the master-elect may be of lower priority. This is reflected in the values for T_{get}(best) by the factor $2T_m$; i.e., the time for two masters to each use the bus for their maximum allotted time. The only buses that do not show this $2T_m$ behavior are the Futurebus and the Nubus.

Futurebus has an intrinsic facility called "preemption," which allows a high-priority master to displace a lower priority master-elect before the bus is handed over. The VME bus has a similar feature provided by the BCLR line, which informs the current master that a high-priority master is waiting to use the bus and which may be used to preempt the master-elect also. But this is not fully specified in the specification, and so it is not included in the comparison figures. The Nubus does not have a priority mechanism for arbitration, so T_{get}(best) and T_{get}(worst) are the same.

T_{get}(worst) is the maximum time after a standard priority module requests the bus in a heavily loaded system, until it receives bus control. This figure includes the fairness-arbitration algorithm, which ensures all requestors get a turn.

Cache. Cache-memory requirements for a bus are often misunderstood. All buses can implement some kind of cache system. The basic categories are instruction caches and data caches. With care, any bus can implement an instruction cache, since the instructions are unlikely to be modified. Data or combined instruction/data caches, are considerably more useful than pure instruction caches, but are more difficult to implement. If a system programmer can define which data is "private" and which data is "shared," he can, with help from a Memory Management Unit, cause the cache system to treat all shared data as "noncacheable." While this is a low-performance, messy, and nontransparent method, all buses can implement caches in this way.

Higher performance systems, which are required to be transparent and/or to cache shared data also, must have some basic facilities built into the bus to maintain consistency.[4] There are two basic consistency schemes possible: write-through, where all write data is written through the cache to the main memory via the bus; and write-back, where written data is simply stored in the cache until it is flushed at a later time. With the write-through system the transaction is visible on the bus and, provided the bus supports a fully handshaken broadcast mechanism for at least the address, other masters having a stale copy of this data can invalidate it (or copy the updated data off the bus as it becomes available).

With the highest performance write-back system, additional information must be stored along with each piece of data to define the ownership of the data. If the data is marked shared, the scheme reverts to write-through; if it is marked exclusive, the write transaction need not appear on the bus. Data becomes shared or exclusive, depending on the status lines on the bus when the data is first read. The data may become invalidated whenever another processor is seen writing the same location over the bus. If a piece of data exists in a cache and is marked exclusive, and if another board wishes to read the location in main memory containing that data, the cache must "intervene" in the read cycle to inhibit the stale data from the main memory and instead provide the master with most recent data from within itself.

Both the Futurebus and the Fastus support an adequate broadcast mechanism to support the write-through scheme fully, but only the Futurebus has the necessary bus status and protocol intervention facilities to support the high-performance write-back scheme.[5]

Measures of cost and complexity. This section of the table was included as an indication of the relative cost effectiveness of each bus. All of the buses are expensive at the moment, but proponents of each claim their bus is less expensive than their competitors. Multibus II may appear expensive to implement at the moment, but this is likely to change. Intel claims that in the near future a crossover point will be reached at which Multibus II will become less costly than, say, the VME bus, because the interface will become more fully integrated and the trend in the cost of silicon is always downward. This does tend to assume that semiconductor manufacturers supporting the VME bus will stand still and not become integrated also—which is hardly likely. However, in the long term, the number of active signal lines is indicative of the ultimate cost effectiveness of the bus interface, since silicon cost predominantly occurs in the interconnections.

A more important question to ask of each bus is how much of the facilities provided must be implemented to use or to gain the principal advantages of the bus. For the VME bus, clearly, cheaper implementations are possible which are

wholly 16-bit data and 24-bit address, although this raises compatibility issues with 32-bit systems. The VME bus also presents a somewhat simple appearance because of its traditional design, and its duplication of the familiar 68000 component-level signals. For the Futurebus and Nubus one must always implement the full 32-bit-wide bus. This means that a minimum cost implementation is less cost-effective if only 16-bit processors are used, than say the VME bus, but it does overcome the compatibility problems and eliminates some complexity for 32-bit processors due to hybrid arrangements. However, the cache facilities, message-passing, broadcast facilities, and sophisticated CSR functions can be eliminated or considerably reduced in cost-conscious implementations of the Futurebus. In particular, boards can be readily constructed using slower but more highly integrated parts such as PALs and still ensure total compatibility because of the technology-independent asynchronous handshake. This makes a wide spectrum of cost and performance possible.

For Multibus II, message passing is necessary to gain any real advantage of the bus over, say, Multibus I because of the clock-latency problem. Message passing is expensive, in LSI, real-estate, and software redevelopment. However, Intel has an ingenious solution to this particular problem. The iSSB serial bus implements the same message-passing function as the parallel bus, so implementations that require low bandwidth and only one or two processors can use just the iSSB bus to become cost-effective.

The philosophy of Nubus is not to burden the user with unnecessary additional features. Nubus therefore has a refreshing sparcity of mechanism, which allows it to be priced considerably cheaper than Multibus II, by virtue of the absence of more sophisticated features. Fastbus tends to be used in very high performance systems, such as those involving the connection of large numbers of data-acquisition subsystems in a tree-structured network, typical of the requirements of the nuclear physics community.

Acknowledgments

I would like to thank Steve Cooper, Shlomo Pri-Tal, George White, and Dave Gustavson for their extensive comments on the first draft of this comparison table.

References

1. H. Kirrmann, "MicroStandards—Report on the Paris Multibus II Meeting," *IEEE Micro,* Vol. 5, No. 4, Aug. 1985, pp. 82-89.
2. R. V. Balakrishnan, "The Proposed IEEE 896 Futurebus—A Solution to the Bus Driving Problem," *IEEE Micro,* Vol. 4, No. 4, Aug. 1984, pp. 23-27.
3. D. B. Gustavson and J. Theus, "Wire-OR Logic on Transmission Lines," *IEEE Micro,* Vol. 3, No. 3, June 1983, pp. 51-55.
4. R. Katz et al., "Implementing a Cache Consistency Protocol," *Proc. 12th Ann. Computer Architecture Conf.,* pp. 276-283. Available from IEEE Computer Society Press.
5. P. Sweazey, "The Futurebus Cacheing System," *Proc. Midcon, Session on Advances in Backplane Bus Technology,* Sep. 1985. Available from Electronic Conventions Inc., 8110 Airport Blvd., Los Angeles, CA 90080.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 150 Medium 151 Low 152

IEEE MICRO

Annual Index Volume 5, 1985

This index covers all technical items that appeared in this periodical during 1985, and items from prior years that were commented upon or corrected in 1985. The index is divided into an Author Index and a Subject Index, both arranged alphabetically.

The *Author Index* contains the primary entry for each item; this entry is listed under the name of the first author and includes coauthor names, title, location of the item, and notice of corrections and comments if any. Cross-references are given from each coauthor name to the name of the corresponding first author. The location of the item is specified by the journal name (abbreviated), year, month, and inclusive pages.

The *Subject Index* contains several entries for each item, each consisting of a subject heading, modifying phrase(s), first author's name, and enough information to locate the item. For coauthors, title, comments, and corrections if any, etc., it is necessary to refer to the primary entry in the Author Index.

Author Index

+ Check author entry for coauthors

† Check author entry for subsequent corrections/comments

Subject Index

Check author entry for coauthors

† *Check author entry for subsequent corrections/comments*

R

S

T

U

W

+ Check author entry for coauthors

† Check author entry for subsequent corrections/comments

MicroReview

Editor: David L. Hannum
AT&T Information Systems

Something to read

This month I received an overwhelming number of items worth bringing to your attention. After sifting through everything, I found it necessary to hold over several of the new arrivals for the coming issues in 1986 so that I could summarize here some of the new books you might be interested in reading.

The Personal Robot Book by Tex Marrs, Tab Books, 1985, $21.95; and

Robotics by Anne Cardoza and Suzee J. Ulk, Tab Books, 1985, $16.95.

These two books on robotics are valuable for nonprofessionals in the field. *The Personal Robot Book* describes, in lay terms, the historical development of robots, giving descriptions of many past and present-day educational, research-oriented, and "labor-saving"(?) robots. These descriptions include some diagrams and many pictures. If you are a beginner in the field or want to interest someone else in robotics, this would be a good starter.

The second book is titled simply *Robotics*. For anyone interested in careers, education, and training in the robotics field, I suggest this book. Although not totally comprehensive of the entire field, this book gives good background information for anyone considering robotics as a career or seeking training in the area to upgrade their skills and knowledge. The authors provide lists of schools and programs throughout the country, other sources of information, and details on career opportunities in robotics.

Next is a book for the writer/engineer who uses Microsoft Word (MS WORD) as the tool for producing tomes.

The Word Book by David Bolocan, Tab Books, 1985, $16.95.

If you want a book that explains a word processor in terms you and I can understand, this is your book. The text is thorough, easy to use, easy to reference, and it provides practical examples that the MS WORD manual does not offer. Some particularly helpful chapters are:

- Chapter 4, Entering a Document. This chapter gets you started using your word processor without the need to read through the whole text—as is necessary with the manual. It starts by tiptoeing through the keyboard, running words, jogging around the screen, stopping at the Command area, and scrolling with the mouse.

- Chapter 5, Editing Text. Here, you can take a ride on the cursor, select text via the function keys, pitch something in the wastebasket and retrieve it, move around sections of a document, and even learn to undo your own accidents (within reason).

- Chapter 7, Formatting Paragraphs. Now, you can continue on to the paragraph workshop, walking through examples of on-screen formatting, direct key formatting, and many more.

- Chapter 12, Window Operation. You can learn to open windows, manipulate information via the windows, and close windows.

- Chapters 16 and 17, Style Sheets. In this chapter, you can troll through the gallery, attach and detach style sheets, or continue your quest for the ideal formatting scheme.

- Chapter 20, Word to Wordstar. Now, you get a break and use conversions of data.

- Chapters 21, 22, and 23, Spell. You will find this is a little utility of unmeasurable value.

This good, complete text lets your fingers do the walking through the mysteries of word processing.

Next month, I plan to present reviews of two new word processors, a color printer (inexpensive), and more. I appreciate all the cards and letters you've been sending. Your comments and suggestions are welcome.

One last note to the person who wrote me about NAPLPS: You're right. The Canadians should get some credit—and are usually ignored. Sorry.

(Ed. note: See Letters to the Editor on page 88 of this issue.)

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card.

High 177 Medium 178 Low 179

DEPARTMENT

MicroLaw

By Richard H. Stern/Law Offices of Richard H. Stern/2101 L Street NW, Suite 800/Washington, DC 20037

This month's column concerns readers' questions that have accumulated over the last several months.

A reader who wants to start up his own computer-related company asks, "How do I avoid getting sued like Steve Jobs did?"

If the new company in any way competes or is likely to compete with the former employer(s) of the startup's principals, one of the first things that the new company can expect is a trade secret or unfair competition lawsuit. There are a number of "dos and don'ts" that any engineer should follow if he has decided to leave his present job and start up a new company. The first of them is not to try following a do-it-yourself schedule like this one, but instead to seek competent legal advice before doing anything—preferably from someone who understands both intellectual property law and the technology involved. Assuming that the reader does not want to take that advice rigorously, or wants at least to sensitize himself to some of the problem areas, I supply the following selection of some "do-don't" highlights:

1. In no circumstances make copies of the employer's documents that are stamped "confidential," "trade secrets," or the like, or that you believe contain information that the employer regards as confidential. Making off with such papers is the act that most frequently causes a trade secret lawsuit against a startup company and the ex-employees. Not only will doing that greatly incense the ex-employer, but it will excite the court against the ex-employees and make it want to adopt a very punitive attitude.

To be sure, some employers stamp everything, including the city telephone directory, confidential. Of course, they are not entitled to legal protection against anyone else's use of commonplace, public-domain material. But if the material is not really a trade secret, why not get your copy of the information from some other source and avoid the possible controversy?

Moreover, a departing engineer should not be under the misimpression that once the employment terminates so, too, does the obligation not to appropriate the trade secrets of the ex-employer; the duty continues. Further, the fact that the departing employee participated in or entirely created the trade-secret information, as part of the employment, does not mean that he can appropriate it for his own profit.

2. Try not to make a wholesale raid of an entire team or department. This also particularly incenses ex-employers, and is likely to cause a suit against the principals of a startup, in which the ex-employer charges them with "conspiring to destroy the business" of the ex-employer or "conspiring to entice away its employees."

In principle, employees (even engineers) are not serfs; they are free to job-hop. Nevertheless, many states take the position that an officer (such as Jobs) or a key employee (such as a manager), so long as he remains an employee, owes a "duty of faithfulness" to his employer, and that it is a breach of fealty for him to solicit coemployees to quit and join a proposed new venture. Therefore, the most cautious thing to do is refrain from discussing employment in the new company with coemployees (or at least from initiating such discussions) while one is still employed at the old company.

Obviously, the fealty principle has limits. It may apply less forcefully to low-ranking employees. And not all courts take the view that this duty is paramount over the employee's interest in economic self-betterment. In one case, for example, an officer and director solicited eight important employees before resigning, they all left together, and the court found no liability.[1] But there are considerable litigation risks involved in approaching coemployees while one is still employed by the old company.

3. If the prior employer has an "exit interview" policy, you should be very careful about what you say at the interview, whether about your future plans or how you feel about the ex-employer. This is a situation where anything you say can and will be used against you, and a prior briefing by a well-informed counsel may be very useful.

In the celebrated "Space Suit" case,[2] during the exit interview the employer's representative treated the employee to some remarks to the effect that "there was a matter of company loyalty and ethics involved." The departing employee angrily responded that "loyalty and ethics had their price," meaning that he had been grossly underpaid for some time. Later, the court stressed that remark as showing that the defendant ex-employee had a bad, disloyal "mental attitude," warranting a punitive response from the judicial system. In this connection, be very careful about signing any acknowledgments that valuable trade secrets of the company have been disclosed to you in the past.

4. The right to take your new-product ideas with you is very problematical. There are potentially both confidentiality and "faithfulness" problems in this situation. The most cautious thing to do is to emulate Wozniak (a more conservative person, legally speaking, than Jobs) when he left Hewlett-Packard to help start up Apple. He told H-P about his plans to leave and to develop a microcomputer, a product in which H-P said it then had no future interest, and they parted amicably. Of course, not every employer has as fair an attitude as H-P, and the parties' respective rights are not always clear.[3]

In one case, the chief engineer and vice president of a valve manufacturer developed an improved valve at home, on weekends and at night, without using company facilities. He did not disclose the product to the company and instead formed a new company to manufacture the valve, in competition against the former employer. The court held that he had violated his fiduciary duty to the

employer by "secreting" the design and forming a new company to exploit it.[4] Three factors of considerable significance in this case, which are not always present, were that the defendant had been a high official of the ex-employer, and therefore had a very strong, "fiduciary" duty (duty of faithfulness) to it; that the product was directly competitive; and that the court regarded the new valve design as a trade secret belonging to the employer.

An interesting question is raised when the original employer company attempts to develop a product, the employee is part of the team working on the project, the employer fails and goes into bankruptcy, the employee takes the product design elsewhere, and he develops it commercially. An argument may be made that the product belongs to the trustee in bankruptcy (and thus, indirectly, to the creditors of the late company), rather than to the ex-employee's new company, so that the ex-employee has misappropriated part of the bankrupt estate.

The whole subject of trade secret suits against startups deserves considerably more discussion, and the foregoing represents only some of the major points to consider. This is a very problematical situation, which requires extreme circumspection, because the risks and potential costs are very high.

A reader asks how to protect printed circuit board layouts under the copyright, semiconductor chip protection, and/or patent laws. Unfortunately, there is no satisfactory, positive answer.

First, the layout of a printed circuit board is probably unprotectable under the copyright laws. The copyright laws do not protect purely utilitarian aspects of product designs, or even products whose ornamental aspects are inseparable from their utilitarian aspects. The unavailability of copyright protection for semiconductor chip layouts led to the enactment of the Semiconductor Chip Protection Act (SCPA), and that strongly indicates the parallel want of copyright protection for printed circuit boards. Some manufacturers of printed circuit boards place copyright notices on their products, but they are probably just whistling in the dark.

The SCPA does not protect printed circuit boards. Its protection is limited to integrated circuits (ICs), or as the SCPA terms them, "semiconductor chip products." SCPA § 901(a)(1) defines semiconductor chip products as multilayer products made up of material deposited onto and etched away from a semiconductor substrate.[5] A printed circuit board does not fit that definition, and the legislative history of the SCPA expressly denies that printed circuit boards are intended to be protected.

Patent protection is theoretically available for printed circuit boards, but in practice it is not. To merit a patent, a printed circuit board layout would have to be inventive—that is, not a routine, obvious development typical of the existing state of technology. That the layout of a printed circuit board may require a great deal of time-consuming, expensive labor would not be enough, in itself, to confer patentability. (In the case of IC layouts and the SCPA, in contrast, that probably would suffice.)

In some circumstances, contractual arrangements and trade secret laws might be used to prevent a customer or supplier from copying a printed circuit board layout. But such arrangements are of doubtful utility in the case of mass-marketed products, since independent third parties and the general public are under no limitations against copying the design.

A reader refers to a cartoon in *EDN* (Feb. 7, 1985, p. 328) that warns: "Early sales talk jeopardizes design rights" under the SCPA. The accompanying text (p. 330) amplifies this point:

> Marketing representatives, as well as engineers, must be careful not to make any presentations that might be construed as sales offers to vendors or OEMs while a semiconductor chip is in the development stage and unregistered with the Copyright Office. Any talk of selling a mask work, or a chip embodying the mask work, could trigger the beginning of the 2-year period in which the mask work must be registered [or else it falls into the public domain]. Such discussions could later be presented as evidence that commercial exploitation had indeed begun. As a result, the design copyrights to a new chip, so carefully crafted by Congress, could be lost.

The reader's question is what to do to avoid the problem and yet go about one's business in the ordinary way. For example, must a manufacturer really silence its salesman?

On this occasion, *EDN* has not quite got its facts right. There was a great deal of technical input from EE's and semiconductor industry personnel when the SCPA was drafted. (See Senator Matthias' MicroView column in *IEEE Micro*, Aug. 1983, pp. 5-6.) Consequently, this potential problem was foreseen and dealt with by an early amendment to the House chip bill. "Talk of selling a mask work, or a chip embodying the mask work" does *not* trigger the beginning of the two-year period within which a layout must be registered to prevent its falling into the public domain.

According to SCPA § 901(a)(5), triggering the two-year period requires one of the following acts:

- selling the chips or transferring (e.g., lending) them by a similar transaction; or
- offering *in writing* to sell the chip, *after* the chip has already been fabricated on a wafer.[6]

Mere talk does not count. Even a letter or other written offer does not count unless the chip has already been fabricated at least once.

The legislative history of the SCPA states that this provision was tailored to meet objections by the semiconductor industry that marketing custom chips would lead to problems if the mere offer to a potential customer to develop them could trigger the two-year clock. The re-

quirement that the offer be in writing, as well, was imposed because the date of written offers can be established by a search of company files or other records, while relying on undocumented salesman's talk as evidence would create business uncertainty of the very type of which the *EDN* article warns.

Incidentally, sale of the mask work, referred to in the passage from *EDN* quoted above, means sale of the layout or design. That will *not* constitute first commercial exploitation for the purposes of starting the SCPA's two-year clock, in any circumstances. Thus, a systems house's sale of a design to a customer, or offer to sell it, even in writing and after wafer fabrication, is not a triggering event. The sale or offer to sell must be of the chips, not of the design for the chips. Finally, *EDN*'s reference to a design copyright in the chips is in error. There is no "design copyright to a new chip, so carefully crafted by Congress." Congress decided not to have chips protected by the copyright law and instead passed a new, industrial property law to govern chip layouts. The mask work right is not a copyright or a patent, but is significantly different from either.

References

1. Sarkes Tarzian, Inc. v. Audio Devices, Inc., 166 F. Supp. 250 (S.D. Cal. 1958), aff'd mem., 283 F.2d 695 (9th Cir. 1960), *cert. denied*, 376 U.S. 869 (1961). A number of similar decisions are collected in the court's decision in Motorola, Inc. v. Fairchild Camera & Instr. Corp., 366 F. Supp. 1173, 1182 (D. Ariz. 1973), the case in which Motorola sued Fairchild for raiding enough major employees to start up a semiconductor company.
2. B.F. Goodrich Co. v. Wohlgemuth, 117 Ohio App. 493, 192 N.E. 2d 99 (1963).
3. Probably, there is a duty to disclose one's plans to leave an employer, prior to actually resigning, only if (1) because of his position in the company, the person has a fiduciary duty (i.e., duty of faithfulness) to the employer, *and* (2) failure to make the disclosure results in a harm to the employer. Thus, Wozniak probably had no duty to disclose his plans, and his conduct was ultraconservative in avoiding difficulties and hard feelings.
4. Daniel Orifice Fitting Co. v. Whalen, 198 Cal. App. 2d 791, 18 Cal. Rptr. 659 (1962).
5. 17 U.S.C. § 901(a)(1) provides: "a 'semiconductor chip product' is the final or intermediate form of any product—
   (A) having two or more layers of metallic, insulating, or semiconductor material deposited on or otherwise placed on, or etched away or otherwise removed from, a piece of semiconductor material in accordance with a predetermined pattern; and
   (B) intended to perform electronic circuitry functions. . . .'
6. 17 U.S.C. § 901(a)(5) provides: "to 'commercially exploit' a mask work is to distribute to the public for commercial purposes a semiconductor chip product embodying the mask work; except that such term includes an offer to sell or transfer a semiconductor chip product only when the offer is in writing and occurs after the mask work is fixed in the semiconductor chip product."

To the Editor:

"What's New?"

To the Editor:

Two articles in recent issues of *IEEE Micro* annoyed me.

MicroReview (October, p. 80) states, "AT&T emerged with a new approach . . . NAPLPS." This is blatantly unfair to the years of work in Canada, sponsored by DOC, before AT&T saw NAPLPS. AT&T made only minor changes.

"Mapping High Level Syntax and Structure Into Assembly Language" (August, p. 67) ignores many related efforts. Its only "new ideas" have been in Intel's ASM-86 for several years. It should not be hard to outdo IASM, so concluding the approach is wrong because their implementation is poor is premature at best.

Ian McIntosh
Scarborough, Ontario, Canada

Author's reply

To the Editor:

I am sorry that Mr. McIntosh was annoyed by our article. I am afraid, however, that I cannot agree that our "poor implementation" led us to a premature conclusion; we used several other authors' high level language assemblers and were even less impressed with these than with our own. I would suggest that Mr. McIntosh implement a high level syntax assembler, use it, and report the results—I am always ready to be convinced by a cogent argument.

I am more concerned that we may have "ignored many related efforts." I discovered, during the time between writing and publication, that the IBM Personal Computer Macro Assembler supports record structures. While not excusing myself, I can only say that extensive research of the literature was carried out (the IBM information was not published, to my knowledge) and that the article was subjected to rigorous refereeing. I would welcome a list of overlooked references from Mr. McIntosh, if these exist.

M. F. Smith
Redditch, Worcestershire, UK

(Editor's note: The author's reply to the first comment appears in this issue at the end of MicroReview.)

New Products

Editor: Kenneth Majithia/IBM Corporation

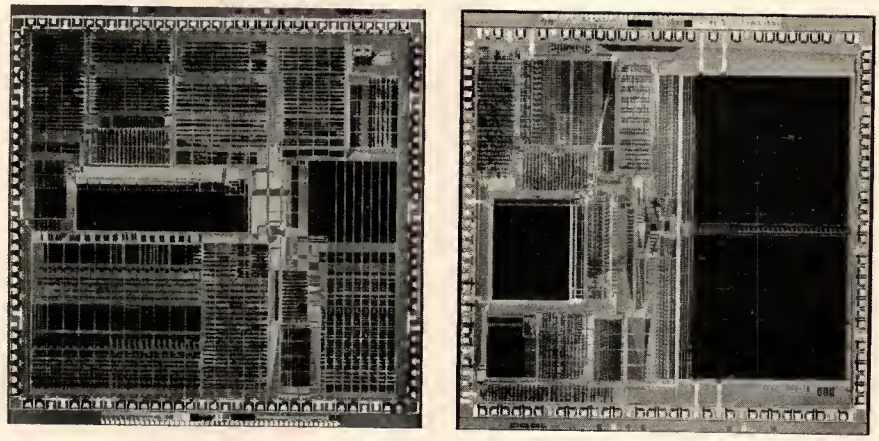
Fairchild enters 32-bit CMOS arena

A CMOS 32-bit microprocessor that executes instructions in 30 nanoseconds—or at an average rate of more than five million per second—has been introduced by Fairchild Camera and Instrument Corp.

Clipper is a general-purpose three-chip module maximized for scientific and professional computing applications in the UNIX operating system environment. Clipper's streamlined instruction set architecture, which runs at 33 MHz, has the basic elements of the RISC (reduced instruction set computer) architecture, coupled with a macroinstruction unit that provides high-level instructions and functions. Hardwired instead of microcoded instructions are used to achieve peak performance levels of up to 33 MIPS.

The module contains a CPU with on-chip floating-point execution unit and two combination cache/ memory-management chips (one each for instructions and data); total transistor count is 846,000. The two cache chips are linked to the CPU via a dual-bus architecture with one 32-bit bus dedicated to instructions, the other to data.

An additional 32-bit, synchronous, multiplexed address/data bus allows the chip set to interface with main memory and with a variety of industry-standard peripheral chips; the flexible bus structure permits byte, half-word, word, and quad-word transfers.



The Fairchild Clipper 32-bit microprocessor is a three-chip set consisting of a CPU (left) and two identical cache/memory-management units (right), one each for instructions and data.

The Clipper CPU chip consists of four major functional blocks: an integer pipe with a three-port 32 × 32 register file, serial 64-bit double-bit shifter, and a 32-bit arithmetic logic unit; a 64-bit floating-point unit with its own register file of eight 64-bit registers; complete prefetch logic to support an eight-byte instruction buffer; and a macroinstruction ROM used to execute sequences of standard machine instructions.

The Clipper module is a 3.0 × 4.5-inch printed circuit card, which plugs into the user's system via a 96-pin connector. Individual chips on the module are packaged in 132-pin ceramic leaded chip carriers.

Priced at $2451.80, Clipper is expected to be available in sample quantities in June 1986 and production volumes in fourth quarter 1986. Initial software offerings are planned to include a port of UNIX System V (Version 2); optimized Fortran, C, and Pascal compilers; and an assembler. A VAX cross-support software package, available immediately, includes an assembler, C compiler, processor simulator, module performance analyzer, debugger, and various utilities.

Contact Fairchild Camera and Instrument Corp. at 464 Ellis St., Drawer No. 7281, Mountain View, CA 94039; (415) 962-5011.

Reader Service Number 31

10-net LAN works over twisted pair wiring

Fox Research, Inc., is offering a high-speed local area network that uses twisted pair wiring for installation.

10-Net is designed for IBM PC, XT, AT, and compatible systems running under DOS 2.0 or a later version. Its addressing and message format are compatible with Ethernet standards.

A dedicated file server is not required with 10-Net. Any station on the network can serve as a "superstation" and share its resources of software and peripherals. The LAN provides print spooling, electronic mail, communication among stations, concurrency control, and multiple security levels.

10-Net nodes (including interface card, tap box, eight-foot cable, software, and manual) are each available for $695; dealer, OEM, and VAR inquiries are invited.

For additional information contact Fox Research, Inc., 7005 Corporate Way, Dayton, OH 45459; (513) 433-2238.

Reader Service Number 32

Users can configure 32-bit board-level computer

National Semiconductor Corporation has introduced a family of 32-bit, board-level computers, called Integrated Computer Modules. The products are based on National's Series 32000 32-bit microprocessor family and are suited for use with workstations, CAD/CAM, robotics, data acquisition, process control, and other computational environments.

The ICM-3216 module makes use of the NS32016 CPU, the NS32081 FPU floating point unit, the NS32082 MMU memory management unit, the NS32201 TCU timing and control unit, and the NS32202 ICU interrupt control unit, along with four serial ports, a parallel port, a Small Computer System Interface, and a 16-bit I/O bus—all of which fit on a 9 × 11-inch board format.

Depending on the desired size of physical memory, the computer module consists of either two or three printed circuit boards (9.18 × 11.02 inches). One is a CPU board that provides computing functions such as CPU, disk/tape control, serial ports, and a printer port. Memory occupies either one or two boards and provides from 1M to 8M bytes of dynamic RAM.

The CPU is National's 10-MHz NS32016 microprocessor, which combines its internal 32-bit architecture with a 16-bit external data path and a 24-bit address bus. This capability provides up to 16M bytes of memory without the burden of segmented addressing.

The second-generation ICM-3232 module is designed to use the NS32032 CPU in its computing cluster and provide the power of the 32-bit data bus. The ICM-3232, scheduled for availability in 1986, is expected to contain 2M bytes of memory on board. It is partitioned into a computing cluster and an I/O processor, which uses the NS32016 under control of the I/O and without the burden of the I/O overhead. I/O processor section is similar to the ICM-3216.

Both the ICM-3216 and the ICM-3232 provide hard-disk capability via a Small Computer Systems Interface. Four RS-232 serial ports, a parallel port, and a real-time calendar clock with battery backup are also provided on board.

The ICM-3216 is priced under $3000 in single-piece quantities. The second-generation products are expected to be priced below $4000. Personality modules are expected to be priced at approximately $1500.

Contact National Semiconductor Corporation at 2900 Semiconductor Drive, Santa Clara, CA 95051; (408) 721-5000.

Reader Service Number 33

Workstation, LAN combine to form plant control system

Triconex Corporation has introduced a family of IBM-compatible color graphics operator workstations and a five-megabit, fault-tolerant local area network. When combined with its Tricon-1 universal controller, the products give users an integrated real-time system for plantwide control.

The Triconex TriView color graphics workstation can be linked to the Tricon-1 universal controller via TriLAN, a token-passing local area network for plantwide fault-tolerant control.

The TriView operator workstation uses the IBM PC AT microcomputer with a color CRT and can be linked to Tricon-1 universal controllers via the TriLAN local area network. TriLAN supports the IEEE 802.4 and GM MAP interconnect standards for token-passing local area networks.

Typical applications for an integrated system include interlock and control of offshore or remote oil production and processing facilities, chemical and petrochemical plants, power plants, and other control applications where process control and safety interlock systems are required.

The TriLAN network allows users to link up to 32 Tricon-1 universal controllers and/or TriView workstations via a high-integrity data path. TriLAN operates transparently to the control system software, uses a token-passing architecture at megabaud transmission rates, and offers multiple levels of redundancy.

TriView and TriLAN are available for purchase now with delivery early in first quarter 1986. For further information on the products and their applications, contact Triconex Corporation, 16800 Aston Street, Irvine, CA 92714; (714) 261-0880.

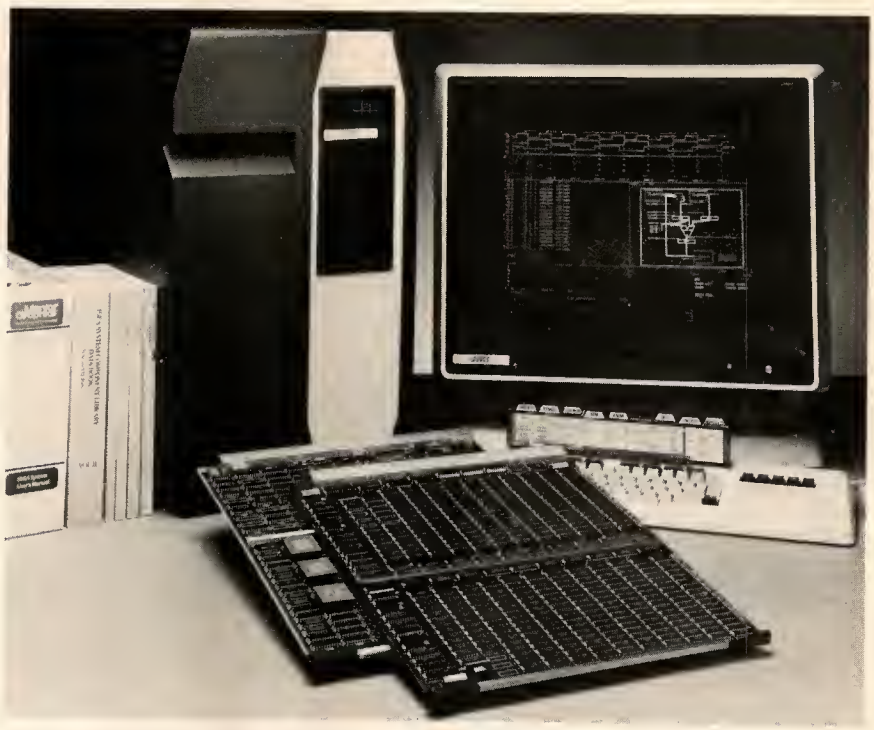
Reader Service Number 34

10-MIPS accelerator tackles a variety of CAE tasks

Mentor Graphics Corporation's general-purpose hardware accelerator, called Compute Engine, is a parallel-processor system that uses Reduced Instruction Set Computer (RISC) concepts and extensive pipelining to produce throughput of 10 million instructions per second. Compute Engine increases throughput on simulation, placement, routing, design-rule checking, and data formatting.

The accelerator's architecture is complemented by a state-of-the-art compiler that converts standard high-level language source code into object code fully optimized for the Compute Engine's parallel-processing environment. Software includes a complete set of program development tools, which allow rapid applications development in a standard HLL.

According to the company, the Compute Engine's logic simulation performance extends to 200,000 gate evaluations per second, depending on the particular parts models being used. Quick-parts models, which condense complex multigate logic functions into a minimum data space, execute at this maximum speed. As a consequence, designs of up to a half million gates (the equivalent of a 32-bit microprocessor) can be simulated directly on the CAE workstation. For conventional parts, speeds approach 100,000 gate evaluations per second and design capacity approaches 100,000 gates. In circuit-level simulations, the Compute Engine has a calculation capacity of 8 million floating-point operations per second.



Mentor Graphics' Compute Engine hardware accelerator optimizes standard C source code, providing a symbolic debugger and an object and image librarian.

Compute Engine, priced at $27,900, includes a CPU board and 4M-byte memory board that fit into the Apollo DN 550 and DN 560 workstations, plus one Mentor Graphics software application upgrade. Both 10M-byte and 20M-byte versions are available for $36,900 and $51,900 respectively. Customers may port their own application software to the Compute Engine by licensing the C compiler and associated program development tools for $150,000. Compute Engine shipments are expected to begin during first quarter 1986.

Mentor Graphics Corporation is located at 9400 Southwest Barnes Road, Suite 140, Portland, OR 97225; (503) 297-1551.

Reader Service Number 35

HP introduces color eight- pen plotter

Hewlett-Packard Company has introduced the HP ColorPro eight-pen plotter supported by 100 software packages. The ColorPro plotter can be combined with a personal computer and graphics software to provide business and presentation graphics.

ColorPro produces multicolor pie, bar, and line graphs and text charts on 8½ × 11-inch (A4/A-size) overhead-transparency film and paper. The plotter's high resolution of 0.001 inch (0.025 mm) allows the plotting of solid areas of color and continuous straight lines.

ColorPro is supported by popular integrated software packages containing graphics, such as 1-2-3 and Symphony from Lotus; dedicated graphics programs, such as Decision Resources' Chart-Master and Sign-Masters; and the HP Graphics Gallery and Textcharts. The plotter also features the HP-GL programming language, which allows users to create programs for specialized business and technical needs.

Compatible with most popular personal computers, the plotter may be purchased with either RS-232-C or IEEE-488 bus interfaces.

The eight-pen ColorPro business plotter (HP 7440) costs $1295; delivery is estimated at stock to two weeks.

For further information, call Hewlett-Packard's sales office listed in the local telephone directory, white pages.

Reader Service Number 36

Compression/expansion processor reduces memory storage requirements

A single-chip device from Advanced Micro Devices, Inc., is designed to speed the incorporation of document compression into office automation equipment, by achieving typical compression ratios of 30:1. The CEP has applications in office network equipment, including workstations, laser printers, copiers, scanners, and facsimile terminals.

The company states that its Am7970 Compression/ Expansion Processor significantly reduces memory storage requirements and transmission time for digitized documents.

The Am7970 CEP compresses and expands two-tone, bit-mapped image data in accordance with internationally accepted CCITT recommendations T.4 and T.6 for groups 3 and 4 facsimile equipment. Depending on the type of document and scanning resolution, the pixel data is compressed five to 50 times; the data can be restored to its original form by the on-chip expansion processor. The compressor and expander can be programmed independently for one-dimensional Huffman coding, two-dimensional read coding, or transparent data transfer.

The CEP operates at a pixel rate of 2 to 8 megabits per second, with a 5-MHz clock. The device can simultaneously expand and compress both text and image data, enabling it to operate in a full-duplex mode. The CEP can be programmed to handle line lengths or document widths of up to 16K pixels.

The Am7970, packaged in a 68-pin ceramic leadless chip carrier, is priced at $245 each in 100-unit quantities.

Advanced Micro Devices can be contacted at 901 Thompson Place, P.O. Box 3453, Sunnyvale, CA 94088; (408) 732-2400.

Reader Service Number 37

Transparent parallel processing speeds number crunching

The 3280MPS multiprocessor system is Perkin-Elmer's 32-bit computer system for transaction processing, data communications, software development, and number crunching applications.

Tightly coupled and asymmetric, the 3280MPS transparent parallel processor is based on a 3280 CPU and supports up to five attached processors. According to the company, any mix of attached processors (IOP/APUs) may be configured to fulfill specific application needs.

Full support of the 3280MPS is provided by OS/32, the company's real-time, multitasking operating system. User tasks are automatically assigned to queues mapped to the CPU and APUs and executed in parallel with full transparency to the user. The XELOS operating system, derived from UNIX System V (Release 5.2), is supported in the uniprocessor configuration. High-level languages available for software development include Fortran VII, Cobol, C, RPGII, Pascal, Coral 66, and Basic II.

In 400 ns, Perkin-Elmer's 3280MPS multiprocessor system can simultaneously complete a single instruction and perform operations on the next three instructions.

The five basic configurations of the 3280MPS system are packaged in a 72-inch cabinet with power supply and AC distribution panels. Each configuration includes 2M- or 4M-byte composite memory modules with integrated controllers and single-bit and double-bit error detection, memory scrubber, and error-logger circuits.

The 3280MPS memory system features multiple CMMs to accomplish memory interleaving, which is expandable to 16M bytes, four-way interleaved. An S-bus, the major communication port to memory, provides two independent 32-bit paths: read and write. It also supports quadword operations to increase the system's data bandwidth.

Price information is available from Perkin-Elmer, Data Systems Group, 2 Crescent Place, Oceanport, NJ 07757; (800) 631-2154.

Reader Service Number 38

CAD/CAM/CAE graphics processor announced

AMF Logic Sciences has introduced the Turbograph 700, a vector-to-raster graphics processor. The Turbograph 700 is a dedicated processor that converts graphics data supplied by a host computer into a format that can be output to a variety of ink-jet, electrostatic, and laser printers and plotters.

Turbograph 700's drawing speed of up to 400,000 vectors per second relieves the host computer of tasks requiring CPU time. The graphics processor allows the user to define graphics elements for custom-tailored output. For example, shaded areas can be filled with patterns selected by the user. Turbograph 700 also features an alphanumeric character generator with a user-definable font library. Other graphics features include variable line styles and opaque or transparent color overlays.

Turbograph 700 input interfaces include RS-232-C 9600-baud serial, Centronics 8-bit parallel, and others.

The desktop Turbograph 700 carries a single-quantity list price of $6995. OEM discounts are available. Delivery is 90 to 120 days ARO.

The company is based at 10808 Fallstone Road, Houston, TX 77099; (713) 879-0536.

Reader Service Number 39

Color printer offers ribbon transfer

Okidata's OKIMATE 20 uses a ribbon-transfer process to create full-color screen dumps on either computer paper or single sheets, as well as near-letter-quality text processing. The printer is offered for use with the IBM-PC and compatibles and the Apple IIe and IIc computers, through the use of its parallel or serial Plug 'N Print interface kits, which are designed for both hardware and software compatibility. Two diskette-based software programs, Learn to Print and Color Screen Print, are included with the printer.

The product will print on a variety of paper, including acetate for instant overhead transparencies in full color. OKIMATE 20's built-in friction and tractor-feed mechanisms will accept both continuous form paper and single sheets.

OKIMATE 20's thumbnail-size, 24-pin print head has a life expectancy of 10 million characters and prints at a speed of 80 characters per second in the draft mode and 40 in letter quality. OKIMATE 20 prints graphics up to 144 by 144 dots per inch.

OKIMATE 20 uses LSI circuitry, custom microchips, high-resolution graphics, and features such as downline-loadable character sets.

The OKIMATE 20 printer and required interface cost $268.

For more information contact Okidata at 532 Fellowship Road, Mt. Laurel, NJ 08054; (609) 235-2600.

Reader Service Number 40

Speech synthesizer supports Commodore 64

Votrax, Inc., has introduced an addition to its Votalker family of speech synthesizers that can speak text automatically as it is entered into a Commodore 64 and can spell words upon command. Votalker C-64's capabilities include a screen echo that allows all words, numbers, punctuation marks, and other symbols to be spoken as they are printed to the terminal screen.

The unit, equipped with the SC-01A speech chip, speaks in unlimited vocabulary and has a Speak command that vocalizes text printed to the screen. Speak can be used with numbers, phrases, and complex expressions. Votalker C-64 also comes equipped with pitch, volume, and rate controls to create more natural-sounding speech.

A Mode command offers users a choice among three types of text vocalization: a conversation mode that reads text in a natural way, with appropriate pauses at punctuation marks; a verbatim mode that reads text and pronounces symbols; and a character mode that spells each word and pronounces numbers and symbols.

Votalker C-64 is priced at $99.95.

For more information, contact Votrax, Inc., 1394 Rankin Road, Troy, MI 48083; (800) 521-1350; in Michigan call collect (313) 588-0341.

Reader Service Number 41

Chip sets offer IBM PC AT compatibility

Chips and Technologies, Inc., founded in January 1985, has announced two initial product offerings.

PC/AT Compatible CHIPSet, whose five chips replace 63 SSI, MSI, and LSI circuits on the IBM PC AT motherboard, is designed for both board-level and OEM systems manufacturers who desire 100-percent AT compatibility with reduced design time and implementation cost. Applications include industrial automation/process control, scientific/medical instrumentation, and engineering workstations, as well as PC AT-compatible personal computers.

The Enhanced Graphics CHIPSet is a four-chip set that replaces the Enhanced Graphics Adapter card used in the IBM PC, PC XT, PC AT, and compatibles. Through higher integration and 256K-DRAM support (in display memory), the Enhanced Graphics CHIPSet reduces the total chip count from 76 to 32. The set is designed for AT add-on board manufacturers as well as OEMs and will run all EGA software. It supports existing video controller standards, including IBM Monochrome and Color Graphics Adapters and the Hercules Adapter Card.

The PC/AT CHIPSet costs $72.40, and the Enhanced Graphics CHIPSet is priced at $85.40 in quantities of 100.

Contact Chips and Technologies, Incorporated, at 521 Cottonwood Drive, Milpitas, CA 95035; (408) 434-0600.

Reader Service Number 42

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Interest Card

High 168
Medium 169 Low 170

Product Summary

Editor: Kenneth Majithia/IBM Corporation
For more information, circle the appropriate Reader Service Number on the Reader Service Card at the back of the magazine.

MANUFACTURER	MODEL	COMMENTS	RS NO.
Chips/Components			
Fairchild Camera and Instrument Corp. 464 Ellis Street Drawer No. 7281 Mountain View, CA 94039 (415) 962-5011	F100415 read/write RAMs	Two 1K × 1 ECL static read/write RAM versions offer 10-ns maximum address access times. The F100415 and F10415 perform high-speed scratchpad, control, and buffer storage. Each includes on-chip address decoding, separate data input and noninverting data output lines, and an active-LOW chip-select line. $6 each in 16-pin DIP packages and $7.50 each in 16-pin quad-sided flatpacks, both in quantities of 1000.	50
Intel Corp. 3065 Bowers Avenue Santa Clara, CA 95051 (408) 987-8080	82502 chip	LSI, CHMOS transceiver device implements the complete set of transmit, receive, and collision-detection functions specified by the IEEE 802.3 10BASE5 and 10BASE2 LAN standards developed for 10M-bps baseband operation. $24 in quantities of 1000.	51
M-tron Industries Inc. PO Box 630 Yankton, SD 57078 (800) 762-8800	SX05050C series	Eight-product series of HCMOS/TTL crystal clock oscillators meet both surface-mount technology and DIP needs. Outputs are TTL-compatible; 19 sample frequencies between 4 and 20 MHz are stocked for fast sample request response. $5.85 each, minimum order of 100.	52
Traveling Software, Inc. 11050 Fifth Avenue N.E. Seattle, WA 98125 (206) 367-8090	Ultimate ROM	Single ROM chip contains the IDEA! outline processor, T-Base database management system, and T-Writer text formatter programs. The chip can be added to Tandy Models 100/200 and NEC PC-8201 lap-sized portable computers. $229.85.	53
Boards			
Peritik Corp. 5550 Redwood Rd. Oakland, CA 94619	Q-bus graphics card series	Ten color and monochrome cards plug into the MicroVax II. One card offers 256 simultaneous colors with 512 × 512 resolution. A monochrome card produces dot graphics plus an alphanumeric overlay with 1024 × 1024 resolution. $615 to $2995.	54
Communications			
Software Research One Natick Executive Park Natick, MA 01760 (617) 655-1133	SNE/FTF network package	File transfer facility connects IBM/MVS, Wang/VS, Digital VAX, and IBM PC systems in information-intensive business environments. Network security is incorporated either through user profiling within SNE or the user of existing security products such as IBM's Resource Access Control Facility. $5000 per minicomputer; $400 per personal computer.	55
Matrix Communication 112-116 Washington Street Marblehead, MA 01945 (617) 639-1211	Alliance PC network	Three-model LAN links up to 20 IBM PCs to share printers, modems, or disks through RS-232 interfaces. The network system is based on a programmable, intelligent cluster controller, which incorporates a multiplex network managed by a Hitachi 64/80 microprocessor. $100 per node.	56
Software			
CAD Design Systems 1305 Remington Rd. Suite D Schaumburg, IL 60195 (312) 882-0114	Design Board Professional	Modeling and design package gives AutoCAD users 3-D CAD capabilities on IBM PC AT/XTs. MegaCADD's software package runs on the 3-D Link Interface and is compatible with AutoCAD peripherals.	57

Calendar

Conferences sponsored or cosponsored by the IEEE Computer Society are indicated by the society's logo. Submit information **eight weeks before cover date** to Calendar, *IEEE Micro*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.

February 1986

ISSCC-86, 33rd IEEE International Solid-State Circuits Conference, February 19-21, Anaheim, California. Contact Lewis Winner, 301 Almeria Ave., Coral Gables, FL 33134, (305) 446-8193 or (305) 446-8194; J. Daneels, Bell Telephone Mfg., Francis Wellesplein 1, 2000 Antwerpen, Belgium, phone 32-37-17 17; or Y. Nishi, Toshiba Co. R&D Center, 1 Komukai Toshiba-Cho, Saiwai-ku, Kawasaki 210, Japan, phone 044-511-2111, ext. 2496.

March 1986

Comdex in Japan 86, March 3-6, Tokyo, Japan. Contact Interface Group, Inc., 300 First Ave., Needham, MA 02194; (617) 449-6600; telex 951176; TWX 710-325-1888.

Compcon Spring 86, March 3-6, San Francisco, California. Contact Glen G. Langdon, Jr., IBM, Dept. K54/282, 5600 Cottle Rd., San Jose, CA 95193; (408) 256-6454 or Compcon Spring 86, IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington DC 20036-1903; (202) 371-0101; TWX 7108250437 IEEECOMPSO.

IEEE Computer Society Built-In Self-Test Workshop, March 12-14, Kiawah Island, South Carolina. Contact Richard Sedmak, Self-Test Services, 6 Lindenwold Terrace, Ambler, PA 19002; (215) 628-9700.

1986 IEEE VLSI Test Workshop, March 18-19, Atlantic City, New Jersey. Contact Gerry Gordon, Philadelphia Electric Co., 133 Fox Chase Lane, Cherry Hill, NJ 08034; (215) 841-4676.

1986 IEEE Workstation Technology and Systems Conference, March 18-20, Atlantic City, New Jersey. Contact Helen Yonan, IEEE Philadelphia Section Office, c/o Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, PA 19101; (215) 898-8106 or (215) 898-8134.

Working Conference on Highly Parallel Computers for Numerical and Signal Processing Applications (IFIP), March 24-26, Nice, France. Contact Michael Barton, Dept. of Electrical Engineering, The University, Bristol BS8 1TR, UK.

April 1986

IEEE Infocom 86, Fifth Annual Joint Conference of the IEEE Computer and Communications Societies, April 7-10, Miami, Florida. Contact IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington DC 20036-1903; (202) 371-0101; TWX 7108250437 IEEECOMPSO.

Tutorial Week Orlando 86, April 7-11, Orlando, Florida. Contact IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington DC 20036-1903; (202) 371-0101; TWX 7108250437 IEEECOMPSO.

ICASSP-86, 1986 IEEE-IECEJ-ASJ International Conference on Acoustics, Speech, and Signal Processing, April 8-11, Tokyo, Japan. Contact Secretariat, ICASSP-86, Simul International, 1-8-10, Akasaka, Minato-ku, Tokyo, 107 Japan; phone (03) 586-8691; telex KYLETYO J23736.

May 1986

CICC-86, Custom Integrated Circuits Conference (IEEE), May 12-14 (educational seminars will be offered May 15), Rochester, New York. Contact Tom Foxall, Pacific Microcircuits, Ltd., 240 H St., PO Box F195-108, Blaine, WA 98230; (604) 536-1886.

Tutorial Week Dallas 86, May 12-16, Dallas, Texas. Contact Martez Camilleri, IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington DC 20036-1903; (202) 371-0101; TWX 7108250437.

Computer Standards Conference 86: Striking a Balance Between Technology, Economics, Politics, and Reality—for Substance, Not Form, May 13-16, San Francisco, California. Contact Stancon 86, IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington DC 20036-1903; (202) 371-0101; TWX 7108250437 IEEECOMPSO.

June 1986

13th Annual International Conference on Computer Architecture (ACM, IPSJ), June 3-5, Tokyo, Japan. Contact IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington DC 20036-1903; (202) 371-0101; TWX 7108250437 IEEECOMPSO.

IEEE Computer Society Tutorial Week Washington 86, June 9-13, Arlington, Virginia. Contact Tutorial Week Washington, IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington DC 20036-1903; (202) 371-0101; TWX 7108250437 IEEECOMPSO.

1986 American Control Conference (AIAA, AIChE, ASME, IEEE, ISA, SCS), June 18-20, Seattle, Washington. Contact Edwin B. Stear, The Washington Technology Center, 376 Loew Hall, FH-10, The University of Washington, Seattle, WA 98195; (206) 545-1920.

23rd ACM/IEEE Design Automation Conference, June 29-July 2, Las Vegas, Nevada. Contact Pat Pistilli, MP Associates, 7366 Old Mill Trail, Suite 101; Boulder, CO 80301; (303) 530-4562.

July 1986

FTCS-16, 16th International Symposium on Fault-Tolerant Computing (IFIP), July 1-3, Vienna, Austria. Contact H. Kopetz, Institut fur Praktische Informatik, Technische Universitat Wien, Gubhausstraße 30/180, A-1040 Vienna, Austria; phone (0222) 56-01.

August 1986

Tutorial Week Chicago 86, August 18-22, Chicago, Illinois. Contact Tutorial Week Chicago 86, IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington DC 20036-1903; (202) 371-0101; TWX 7108250437 IEEECOMPSO.

September 1986

Eusipco 86, European Signal Processing Conference, September 2-5, The Hague, The Netherlands. Contact Eusipco 86 Conference Secretariat, Dept. of Applied Physics, Room 236, Delft University of Technology, PO Box 5046, NL-2600 GA Delft, The Netherlands; phone 31-15-78-1416; telex 38151-bhthd-nl.

International Test Conference, September 8-11, Washington DC. Contact Doris Thomas, PO Box 264, Mount Freedom, NJ 07970; (201) 895-5260.

Tutorial Week Boston 86, September 15-19, Boston, Massachusetts. Contact Tutorial Week Boston 86, IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington DC 20036-1903; (202) 371-0101; TWX 7108250437 IEEECOMPSO.

Iecon 86, Industrial Applications of Mini, Micro, and Personal Computers (IEEE), September 29-October 3, Milwaukee, Wisconsin. Contact Guy O. Beale, Vanderbilt University, Box 1698, Station B, Nashville, TN 37235; (615) 322-2212.

Advertisers

FOR DISPLAY ADVERTISING INFORMATION CONTACT

Southern California and Mountain States: Richard C. Faust Company, 24050 Madison Street, Suite 100, Torrance, CA 90505; (213) 373-9604.
Northern California and Pacific Northwest: Don Farris Company, 161 W. 25th Ave., #102B, San Mateo, CA 94403; (415) 349-2222.
Jack Vance, P.O. Box 3205, Saratoga, CA 95070, (408) 741-0354
East Coast: Hart Associates, Inc., P.O. Box 339, 42 Lake Blvd., Matawan, NJ 07747; (201) 583-8500.
New England: Arpin Associates, P.O. Box 227, Weston, MA 02193; (617) 899-5613.
George Watts, III, 4 Conifer Dr., Wilbraham, MA 01095; (413) 596-4747.
Midwest: Thomas Knorr, Knorr MicroMedia, Inc., 333 North Michigan Ave., Chicago, IL 60601; (312) 726-2633.
Southeast: Larry C. Shattles, 133 Laurel Oak Drive, Longwood, FL 32779; (305) 788-1950.
Southwest: The House Company, 3817 Richmond Avenue, Suite 110, Houston, TX 77027; (713) 622-2868.

Advertising Manager: Mike Koehler, *IEEE MICRO*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720, (714) 821-3240, 821-8380,

For production information, conference or classified advertising contact Sandra J. Arteaga, *IEEE MICRO*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720, (714) 821-1140.

Products

IEEE MICRO

For further information on advertised products, new products, or literature, fill out the **Reader Service Card** (top). Circle the number on the RS Card that corresponds to the number of the item for which you would like more information.

To indicate your interest in an article or department, fill out the **Reader Interest Card** (bottom). Circle the number on the RI Card that corresponds to the level of interest given in the Reader Interest Survey at the end of the article or department.

Please print or type your name and address.

READER SERVICE CARD

IEEE MICRO INFORMATION ABOUT PRODUCTS

Void after June 30,1986

12/85

Name _____

Company _____

Address _____

City _____

State _____

Zip _____

Country _____

Title _____

Telephone number ( ) _____

PRODUCTS PURCHASED OR SPECIFIED	FOR JOB	FOR HOBBY
Computers	☐	☐
Peripherals	☐	☐
Data communications equip.	☐	☐
Memories, components	☐	☐
Software and services	☐	☐
Publications	☐	☐
Other	☐	☐

☐ Please send me information on advertising in *IEEE Micro*.

Product announcements, and products for review, should be sent to Marie English, Managing Editor, *IEEE Micro*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.

Send more information on numbered items:

1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61	65	69	73	77	81	85	89	93	97
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62	66	70	74	78	82	86	90	94	98
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63	67	71	75	79	83	87	91	95	99
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80	84	88	92	96	100

READER INTEREST CARD

IEEE MICRO EDITORIAL RESPONSE

12/85

Name _____

Company _____

Address _____

City _____

State _____

Zip _____

Country _____

Title _____

Telephone number ( ) _____

Comments:

What trends and developments in 32-bit microprocessor technology should we cover in future issues?

Continue comments on other side

Reader Interest Survey:

101	116	131	146	161	176	191
102	117	132	147	162	177	192
103	118	133	148	163	178	193
104	119	134	149	164	179	194
105	120	135	150	165	180	195
106	121	136	151	166	181	196
107	122	137	152	167	182	197
108	123	138	153	168	183	198
109	124	139	154	169	184	199
110	125	140	155	170	185	200
111	126	141	156	171	186	201
112	127	142	157	172	187	202
113	128	143	158	173	188	203
114	129	144	159	174	189	204
115	130	145	160	175	190	205

☐ Please send me the IEEE-CS *Publications Catalog*.

☐ Please send me an IEEE Fellow nomination form.

☐ Please send me an IEEE Computer Society membership application.

☐ Please send me an *IEEE Micro* author's guide.

☐ Please send me information about reviewing article submissions for *IEEE Micro*.

This PO box for reader service cards only.

PLACE STAMP HERE

IEEE MICRO

Reader Service Inquiries
Box 24168
Los Angeles, CA 90024
USA

Comments on articles and other editorial matter:

I liked _____

I disliked _____

I would like _____

For Reader Interest Survey, see other side

PLACE STAMP HERE

IEEE MICRO

10662 Los Vaqueros Circle
Los Alamitos, CA 90720-2578
USA

CALL FOR PAPERS

IEEE MICRO

IEEE MICRO invites its readers to submit articles for three of its 1986 special issues:

APRIL ISSUE
32-Bit Peripheral Chips
Deadline: February 1, 1986

JUNE ISSUE
Telecommunications and Networking
Deadline: February 15, 1986

AUGUST ISSUE
Operating Systems
Deadline: March 15, 1986

Send submissions to:
James J. Farrell III
Editor-in-Chief, IEEE MICRO
c/o VLSI Technology Incorporated
10220 South 51st Street
Phoenix, AZ 85044
(602) 893-8574

Articles will focus on the technological and economic issues surrounding these topics. Tutorials are welcome.

Please include six copies of your manuscript, or contact the Managing Editor at (714) 821-8380 for information regarding electronic manuscript submission.

Do you need further information on submitting articles to IEEE MICRO? Is there a special topic that you want IEEE MICRO to cover in depth? Fill out the Reader Interest Card at the back of the magazine.